

REPORT DOCUMENTATION PAGE			Form Approved OMB NO. 0704-0188		
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA, 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.					
1. REPORT DATE (DD-MM-YYYY) 20-12-2016		2. REPORT TYPE Final Report		3. DATES COVERED (From - To) 1-May-2013 - 30-Sep-2016	
4. TITLE AND SUBTITLE Final Report: Simulation of fluid-structure interaction for high-Reynolds-number compressible flow			5a. CONTRACT NUMBER W911NF-13-1-0146		
			5b. GRANT NUMBER		
			5c. PROGRAM ELEMENT NUMBER 206022		
6. AUTHORS			5d. PROJECT NUMBER		
			5e. TASK NUMBER		
			5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAMES AND ADDRESSES Florida A&M University 1700 Lee Hall Drive 400 FHAC Tallahassee, FL 32307 -3200			8. PERFORMING ORGANIZATION REPORT NUMBER		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS (ES) U.S. Army Research Office P.O. Box 12211 Research Triangle Park, NC 27709-2211			10. SPONSOR/MONITOR'S ACRONYM(S) ARO		
			11. SPONSOR/MONITOR'S REPORT NUMBER(S) 62900-EG-REP.7		
12. DISTRIBUTION AVAILABILITY STATEMENT Approved for Public Release; Distribution Unlimited					
13. SUPPLEMENTARY NOTES The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy or decision, unless so designated by other documentation.					
14. ABSTRACT The objective of the proposed research was to develop a computational framework to simulate complex compressible flow with fluid-structure interaction. To achieve this goal, we developed an immersed boundary method with capabilities of adaptive mesh refinement to simulate flows around bodies of arbitrary geometry within a novel energy conservative compressible flow solver algorithm. Furthermore, we have incorporated the capability to couple the effect of structural dynamics with the flow field. The code is written in a flexible manner to allow different types of fluid-structure interaction by interchanging the module for capturing the structural dynamics. We					
15. SUBJECT TERMS Computational Fluid Dynamics, Structural Mechanics, Fluid-Structure Interaction					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON Kunihiko Taira
a. REPORT UU	b. ABSTRACT UU	c. THIS PAGE UU			19b. TELEPHONE NUMBER 850-645-0140

Report Title

Final Report: Simulation of fluid-structure interaction for high-Reynolds-number compressible flow

ABSTRACT

The objective of the proposed research was to develop a computational framework to simulate complex compressible flow with fluid-structure interaction. To achieve this goal, we developed an immersed boundary method with capabilities of adaptive mesh refinement to simulate flows around bodies of arbitrary geometry within a novel energy conservative compressible flow solver algorithm. Furthermore, we have incorporated the capability to couple the effect of structural dynamics with the flow field. The code is written in a flexible manner to allow different types of fluid-structure interaction by interchanging the module for capturing the structural dynamics. We also comment on ongoing and future extensions of the code.

Enter List of papers submitted or published that acknowledge ARO support from the start of the project to the date of this printing. List the papers, including journal references, in the following categories:

(a) Papers published in peer-reviewed journals (N/A for none)

Received

Paper

08/30/2015	1.00	Ralph Smith, William Oates, Michael Hays, Paul Miles. Bayesian uncertainty analysis of finite deformation viscoelasticity, Mechanics of Materials, (12 2015): 0. doi: 10.1016/j.mechmat.2015.07.002
------------	------	---

TOTAL: 1

Number of Papers published in peer-reviewed journals:

(b) Papers published in non-peer-reviewed journals (N/A for none)

Received

Paper

TOTAL:

Number of Papers published in non peer-reviewed journals:

(c) Presentations

- M. Jemison, "A Collocated, Energy-Preserving Method for Low-Mach Number Compressible Flow, with Fluid Structure Interaction," 40th SIAM Southeastern Atlantic Section Conference (SIAM-SEAS), Athens, GA, March 12-13, 2016.

- M. Jemison and K. Taira, "An Energy Conservative, Skew-Symmetric Method for Compressible Flow with Adaptive Mesh Refinement and Fluid-Structure Interaction," 2016 SIAM Annual Meeting, Boston, MA, July 11-15, 2016.

Number of Presentations: 2.00

Non Peer-Reviewed Conference Proceeding publications (other than abstracts):

<u>Received</u>	<u>Paper</u>
12/20/2016 6.00	Paul Miles, Adriane Guettler, Yousuff Hussaini, William S. Oates. Uncertainty Analysis of Dielectric Elastomer Membranes Under Multi-Axial Loading, ASME 2015 Conference on Smart Materials, Adaptive Structures and Intelligent Systems. 20-DEC-16, Colorado Springs, Colorado, USA. : ,
12/20/2016 5.00	Paul Miles, Michael Hays, Ralph Smith, William S. Oates. Uncertainty Analysis of a Finite Deformation Viscoelastic Model, ASME 2014 Conference on Smart Materials, Adaptive Structures and Intelligent Systems. 08-SEP-14, Newport, Rhode Island, USA. : ,
TOTAL:	2

Number of Non Peer-Reviewed Conference Proceeding publications (other than abstracts):

Peer-Reviewed Conference Proceeding publications (other than abstracts):

<u>Received</u>	<u>Paper</u>
TOTAL:	

Number of Peer-Reviewed Conference Proceeding publications (other than abstracts):

(d) Manuscripts

<u>Received</u>	<u>Paper</u>
TOTAL:	

Number of Manuscripts:

Books

Received Book

TOTAL:

Received Book Chapter

TOTAL:

Patents Submitted

Patents Awarded

Awards

Graduate Students

NAME	PERCENT SUPPORTED	Discipline
Paul Miles	1.00	
Ryan Jantzen	0.30	
FTE Equivalent:	1.30	
Total Number:	2	

Names of Post Doctorates

NAME	PERCENT SUPPORTED
Matthew Jemison	0.70
FTE Equivalent:	0.70
Total Number:	1

Names of Faculty Supported

<u>NAME</u>	<u>PERCENT SUPPORTED</u>	National Academy Member
William Oates	0.08	
Kunihiko Taira	0.08	
FTE Equivalent:	0.16	
Total Number:	2	

Names of Under Graduate students supported

<u>NAME</u>	<u>PERCENT SUPPORTED</u>	Discipline
Daniel Canuto	0.20	Bachelor of Science
Peter Van Brussel	0.10	Bachelor of Science
FTE Equivalent:	0.30	
Total Number:	2	

Student Metrics

This section only applies to graduating undergraduates supported by this agreement in this reporting period

The number of undergraduates funded by this agreement who graduated during this period: 2.00

The number of undergraduates funded by this agreement who graduated during this period with a degree in science, mathematics, engineering, or technology fields:..... 2.00

The number of undergraduates funded by your agreement who graduated during this period and will continue to pursue a graduate or Ph.D. degree in science, mathematics, engineering, or technology fields:..... 2.00

Number of graduating undergraduates who achieved a 3.5 GPA to 4.0 (4.0 max scale):..... 2.00

Number of graduating undergraduates funded by a DoD funded Center of Excellence grant for Education, Research and Engineering:..... 0.00

The number of undergraduates funded by your agreement who graduated during this period and intend to work for the Department of Defense 0.00

The number of undergraduates funded by your agreement who graduated during this period and will receive scholarships or fellowships for further studies in science, mathematics, engineering or technology fields:..... 0.00

Names of Personnel receiving masters degrees

<u>NAME</u>
Ryan Jantzen
Total Number:

1

Names of personnel receiving PhDs

<u>NAME</u>
Total Number:

Names of other research staff

<u>NAME</u>	<u>PERCENT SUPPORTED</u>
FTE Equivalent:	
Total Number:	

Sub Contractors (DD882)

1 a. Florida State University

1 b. Sponsored Research Administration

874 Traditions Way, Third Floor

Tallahassee FL 323064166

Sub Contractor Numbers (c):

Patent Clause Number (d-1):

Patent Date (d-2):

Work Description (e):

Sub Contract Award Date (f-1):

Sub Contract Est Completion Date(f-2):

1 a. Florida State University

1 b. 97 South Woodward Avenue, Third

Tallahassee FL 323060001

Sub Contractor Numbers (c):

Patent Clause Number (d-1):

Patent Date (d-2):

Work Description (e):

Sub Contract Award Date (f-1):

Sub Contract Est Completion Date(f-2):

Inventions (DD882)

Scientific Progress

See Attachment

Technology Transfer

Final Report prepared for the Army Research Office
Program Manager: Dr. Matthew Munson (Fluid Dynamics)

Grant: W911NF-13-1-0146

(compiled on 20 December 2016)

Simulation of Fluid-Structure Interaction for High-Reynolds-Number Compressible Flow

Kunihiko Taira[†] and William S. Oates[‡]

Department of Mechanical Engineering
Florida Center for Advanced Aero Propulsion
Florida A&M/Florida State University, Tallahassee, FL 32310

The objective of the proposed research was to develop a computational framework to simulate complex compressible flow with fluid-structure interaction. To achieve this goal, we developed an immersed boundary method with capabilities of adaptive mesh refinement to simulate flows around bodies of arbitrary geometry within a novel energy conservative compressible flow solver algorithm. Furthermore, we have incorporated the capability to couple the effect of structural dynamics with the flow field. The code is written in a flexible manner to allow different types of fluid-structure interaction by interchanging the module for capturing the structural dynamics. We also comment on ongoing and future extensions of the code.

CONTENTS

1. Introduction	2
1.1. Compressible Flow Solver	2
1.2. Structural Mechanics Solver	4
1.3. Fluid-Structure Interaction	4
2. Compressible Flow Solver	5
2.1. Finite Difference Discretization	6
2.2. Discrete Operators	6
2.3. Semi-Discrete Second Order Finite Difference Formulation	7
2.4. Turbulence Model	9
3. Dynamic Block-Structured Adaptive Mesh Formulation	9
3.1. Thermodynamic Variable Choice	10
3.2. Staggered vs. Collocated AMR Methods	11
4. Immersed Boundary Formulation	11
4.1. Characteristic Propagation Method	12
4.2. Adaptive Characteristic Propagation Method	16
5. Solid Mechanics Model Development	17
5.1. Viscoelastic and Electrostrictive Constitutive Modeling	17
5.2. Electrostriction of Membranes	18

[†] PI, ktaira@fsu.edu, kunihiko.taira@fam.u.edu

[‡] co-PI, woates@fsu.edu

5.3. Bayesian Statistical Analysis	21
6. Structure Code Development	21
6.1. Dynamic Equation	22
6.2. Approximate Solution	23
7. Fluid-Structure Interactions	23
8. Results	23
8.1. Compressible Viscous Vortex	24
8.2. Two-Dimensional Turbulence in Bi-Periodic Domain	24
8.3. 3D Taylor–Green Vortex/Turbulence	27
8.4. Direct Numerical Simulation of 3D Isotropic Turbulence	30
8.5. Structural Mechanics	31
8.6. Fluid-Structure Interaction	32
9. Resulting Publications	33
10. Participants	34
11. Remarks and Future Work	34
12. Acknowledgements	35
Appendix A	35
Appendix B	41

1. Introduction

1.1. Compressible Flow Solver

The study of computational fluid dynamics allows scientists across disciplines to simulate events and gather data that would otherwise be too expensive or impossible to obtain. Of particular interest is the interaction of fluid flow with solid bodies, such as airfoils (Johnson *et al.* 2005; Rhie & Chow 1983; Stevens & Lewis 2003), ship hulls (Dommermuth *et al.* 2014; Faltinsen 1993; Peri *et al.* 2001), microfluidics devices (Erickson 2005; Erickson & Li 2004; Karniadakis *et al.* 2006), and micro-air vehicles (Mueller 2001; Pines & Bohorquez 2006; Ramamurti *et al.* 2000). The phenomena associated with fluid/structure interaction can take place over a range of length-scales and Reynolds numbers, with special consideration that must be taken in the turbulent flow regime (Pope 2000; Schlichting *et al.* 2000; Tritton 2012; White 1991). As we seek to model and simulate these events, we require great computational resources to complete these tasks. It is desirable that we balance the need for fidelity of the simulation with the requirement that the method is computationally tractable. In this work, we will focus on the conservation of energy while performing computations on structured grids, in the context of an adaptive mesh hierarchy. This allows us to focus the majority of the computational effort in the regions of interest, while discretely satisfying conservation of mass and energy.

Morinishi *et al.* (1998) presents both a fully-discrete and semi-discrete method for simulating the incompressible Navier-Stokes equations, while retaining secondary conservation. A framework was presented to derive schemes of arbitrary spatial order. The property of secondary conservation allows for accurate computation of large eddy simulations, due to conservative transport of kinetic energy.

Work by Wall *et al.* (2002) uses a semi-implicit Crank-Nicholson time integrator to achieve second order conservation of kinetic energy for variable density flow, extending the earlier work of Pierce (2001) on the compressible Navier-Stokes equations. Honein & Moin (2004) achieve numerical stability by using non-dissipative, high-order methods

that satisfy various standard thermodynamic properties in semi-discrete fashion, in conjunction with a skew-symmetric convective derivative. A discussion of various skew-symmetric convective discretizations is included in their work on compressible flow.

A different form of the skew-symmetric convective operator is used by [Pantano *et al.* \(2007\)](#) in Large Eddy Simulations on patch-based refined meshes. A tuned centered differencing (TCD) and weighted essentially non-oscillatory (WENO) scheme is used to suppress spurious oscillations caused by dispersion errors at coarse/fine mesh boundaries, allowing for shock-capturing. However, the skew-symmetric form used in [Pantano *et al.* \(2007\)](#) is noted by [Honein & Moin \(2004\)](#) to suffer from a lack of robustness.

[Desjardins *et al.* \(2008\)](#) proposes a method for extending ([Morinishi *et al.* 1998](#)) to variable-density flows in complex geometries, with cylindrical or non-uniform Cartesian meshes. A framework for developing such schemes with arbitrary spatial accuracy was presented, resulting in good agreement with the kinetic energy spectra for homogeneous isotropic turbulence simulations. However, the discretization does not satisfy discrete secondary conservation ([Morinishi 2010](#)).

The work of [Kramer *et al.* \(2009\)](#) seeks to achieve energy stability for non-dissipative schemes in the context of patch-based adaptive mesh refinement. This method uses a summation-by-parts criterion to ensure stability at coarse/fine mesh boundaries, at the expense of using complex stencils that are dependent on the structure of the mesh refinement.

[Morinishi \(2010\)](#) develops both a fully-discrete and semi-discrete formulation of the compressible Navier-Stokes equations for low-Mach number flow using a skew-symmetric convective form that satisfies secondary conservation. The fully-discrete formulation simulates mass, momentum, and internal energy, while maintaining total energy conservation through the spatio-temporal discretization. It is demonstrated that total energy conservation for the semi-discrete formulation is proportional to the time step, rather than spatial order or resolution, with convergence of total energy conservation shown to be of the order of the time integrator. [Brouwer *et al.* \(2014\)](#) develop a class of time integrators of arbitrary accuracy for skew-symmetric methods for the compressible Navier-Stokes equations. [Subbareddy & Candler \(2009\)](#) derive a fully discrete, finite volume scheme that yields a consistent kinetic energy flux. This is done using a specially-derived density-weighted time integrator.

The simulation of interaction between fluid flow and solid bodies has been pursued for some time. One of the first such techniques was the Immersed Boundary Method (IBM). The general concept behind the IBM is to represent a solid body using a set of Lagrangian nodes, while performing fluid calculations on a non-conforming Eulerian grid. The forces associated with fluid/structure interaction at the interface are transferred from the Lagrangian nodes to the Eulerian grid through the use of a discrete Dirac delta function. The original work on the IBM by [Peskin \(1977\)](#) sought to simulate the beating heart, computing the motion of the valves of the heart and the fluid flow within. An extensive body of literature exists surrounding the IBM. A thorough review may be found in [Peskin \(2002\)](#) or [Mittal & Iaccarino \(2005\)](#).

Embedded boundary/geometry methods allow for a fluid stencil to be applied uniformly throughout the domain in the presence of a solid body. The solid is accounted for by altering flow variables in the vicinity of the fluid/solid interface, or by using a forcing term. Recent work deals with large-eddy simulation of turbulent flows past complex or moving boundaries ([Yang & Balaras 2006](#)), tracking the forcing applied at fluid cells recently uncovered by the moving solid. The method of [Udaykumar *et al.* \(2001\)](#) fills vacated cells using an interpolant calculated from surrounding fluid values. The embedded boundary method of [Mittal *et al.* \(2008\)](#) fills “ghost cells” in a narrow band near the

fluid/structure interface for the purpose of applying boundary conditions. A normal probe is extended from the ghost cell centroid to the interface, then into the fluid. The value in the ghost cell is then determined by interpolating from the values surrounding the normal probe. Calculating the ghost values may be non-trivial, depending on how the solid is cut by the grid, as the linear systems posed by the interpolation problem may be coupled between ghost cells.

The Immersed Boundary Method is extended to an adaptive grid hierarchy in [Roma *et al.* \(1999\)](#) and to a formally second-order accurate version in [Griffith *et al.* \(2007\)](#). Other work ([Kalitzin & Iaccarino 2003](#); [Vanella & Balaras 2009](#); [Vanella *et al.* 2008, 2010](#)) has also examined the IBM in the context of adaptive mesh refinement, with applications to moving interfaces or turbulent LES. These methods all share the same constraint that the solid body, or at least the fluid/solid interface, must be entirely contained on the finest level of mesh refinement.

1.2. Structural Mechanics Solver

The utilization of both functional materials (i.e., smart materials) and adaptive structures is known to provide unique capabilities for real time control of a structure's shape, stiffness, or damping which is relevant to many flow control applications ([Smith 2005](#)). Knowledge of the viscoelastic constitutive behavior over a broad range of deformation rates becomes particularly important in many applications where dynamic tunability or actuator-based flow control is of critical importance. Thus we have focused on rate dependent material evolution and the quantification and analysis of model parameter uncertainty to provide critical analysis to fluid-structure codes that exhibit complex dynamic excitations. The following discussion briefly describes the theory followed by numerical implementation and experimental validation.

In addition to viscoelastic constitutive model development is beam, plate, and nonlinear shell code development for integration into flow solvers that couple the structure and fluid via the immersed boundary layer method. Prior Matlab code based on theory and numerical methods summarized elsewhere ([Smith 2005](#)) was converted to Fortran code. Prior codes have been implemented by Oates for use in nonlinear control of beams and plates using smart materials ([Oates *et al.* 2009](#); [Oates & Smith 2008](#)) which were selected for their efficiency for fluid-structure problems.

1.3. Fluid-Structure Interaction

Ideally, the fluid/structure interaction problem could be solved on a single grid with uniform mesh resolution. However, the difference between the characteristic thickness of the boundary layer and the domain size ([Schlichting *et al.* 2000](#); [White 1991](#)) often makes this infeasible. Conformal grid-based methods for fluid/structure interaction, such as finite volume ([Anderson & Bonhaus 1994](#); [Breuer 1998](#); [Jameson & Mavriplis 1986](#); [Khalighi *et al.* 2010](#); [Kim & Choi 2000](#); [Norris *et al.* 2011](#)), finite element ([Brooks & Hughes 1982](#); [Martineau & Berry 2004](#); [Mittal & Tezduyar 1994](#); [Tezduyar *et al.* 2010](#)), spectral difference ([Liu *et al.* 2006](#); [Mohammad *et al.* 2010](#); [Yu *et al.* 2011, 2014](#)), and spectral element methods ([Blackburn & Henderson 1996](#); [Henderson & Karniadakis 1995](#); [Karniadakis & Sherwin 2013](#); [Winters & Kopriva 2014](#)), use meshes such that the boundaries of cells or elements conform to the surface of the solid. This allows for a computational mesh that has enough resolution to accurately resolve flow near the body but lower resolution in the far-field. This saves computational time by focusing computations on the regions that are relevant to the problem at hand. These methods typically avoid the expensive procedures associated with immersed boundary or immersed

interface methods, which involve applying the appropriate boundary conditions at the fluid/solid interface. However, these conformal grid methods may incur significant cost in the presence of moving or deforming bodies, which require remeshing (rezoning) or recalculation of the transfinite mapping parameters (Ferziger & Peric 1999; Tezduyar 2001). Work has also been done to develop unstructured grid methods that use elements that do not conform with the immersed boundary (Ilinca & Hétu 2011; Zhang *et al.* 2011b).

In this work, a methodology is developed for extending the collocated, energy-conserving scheme of Morinishi (2010) to a multi-level grid hierarchy. A new conservative, collocated tensor discretization, which ensures simultaneous consistency with the total, internal, and kinetic energy equations, is presented. The exact conservation of mass and total energy implied by the governing equations is demonstrated in the numerical method. Additionally, a novel method for implementing fluid/structure interaction in a multi-level grid hierarchy, which allows for levels of mesh refinement to cut across the interface, is presented.

2. Compressible Flow Solver

The governing equations for compressible fluid flow are the Navier-Stokes equations (which include the continuity equation, as well as equations for momentum and energy). The system is closed with an equation of state, a constitutive relation which relates the state variables density, pressure, and energy. Assuming a general convective derivative, as well as constant viscosity (μ) and thermal diffusivity (κ), these equations can be written as

$$\begin{aligned}\frac{\partial \rho}{\partial t} + \frac{\partial u_i}{\partial x_i} &= 0 \\ (Conv.)_{u_i} + \frac{\partial p}{\partial x_i} &= \frac{\partial \tau_{ij}}{\partial x_j} \\ (Conv.)_E + \frac{\partial (p u_i)}{\partial x_i} &= \frac{\partial (\tau_{ij} u_i)}{\partial x_j} - \frac{\partial q_j}{\partial x_j},\end{aligned}\tag{2.1}$$

where

$$\tau_{ij} = \mu \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) - \frac{2}{3} \mu \frac{\partial u_k}{\partial x_k} \delta_{ij}, \quad q_j = -\kappa \frac{\partial T}{\partial x_j}\tag{2.2}$$

Here, let δ_{ij} be the Kronecker delta function. Assuming that the fluid behaves as an ideal gas, then pressure can be written in the form of an equation of state, as a function of density and internal energy. In this context, E and e_{int} refer to specific energy and specific internal energy, i.e. energy per unit mass. This will be simply referred to as “energy” for the remainder of this work.

$$E = e_{int} + \frac{1}{2} u_i u_i, \quad p(\rho, e_{int}) = (\gamma - 1) \rho e_{int}\tag{2.3}$$

Parameter $\gamma \equiv C_p/C_v$ is the ratio of specific heat, where C_p and C_v are the specific heat at constant pressure and volume, respectively. The Prandtl number $Pr \equiv C_p \mu / \kappa$ is the ratio of momentum diffusion to thermal diffusion.

Here, $(Conv.)$ represents a general convective derivative, whether in conservative, non-conservative, or skew-symmetric form. The subscript represents the variable on which the operator acts. So, $(Conv.)_{u_i}$ represents the convective operator for the i^{th} component of momentum, and $(Conv.)_E$ represents the convective operator for total energy. The form of the convective operator has significance for the conservation properties of the

method, whether conserving mass, momentum, or energy. The convective operator for some quantity φ can take the following forms:

$$\begin{aligned} (Div.)_\varphi &= \frac{\partial(\rho\varphi)}{\partial t} + \frac{\partial(\rho\varphi u_j)}{\partial x_j} \\ (Adv.)_\varphi &= \frac{\partial(\rho\varphi)}{\partial t} + \rho u_j \frac{\partial\varphi}{\partial x_j} \\ (Skew)_\varphi &= \sqrt{\rho} \frac{\partial(\sqrt{\rho}\varphi)}{\partial t} + \frac{1}{2} \left(\rho u_j \frac{\partial\varphi}{\partial x_j} + \frac{\partial(\rho\varphi u_j)}{\partial x_j} \right) \end{aligned} \quad (2.4)$$

If the divergence form of the convective operator is utilized for the momentum equations, momentum conservation is enforced. However, it has been found for applications to Large Eddy Simulations (LES) that explicit enforcement of momentum conservation can result in adverse effects on the transfer between internal and kinetic energy (Morinishi 2010; Morinishi *et al.* 1998). Suppose, instead, that the skew-symmetric form is utilized for momentum convection. Then, an expression for transport of kinetic energy transport can be obtained by multiplying $(Skew)_{u_i}$ by velocity component u_i and summing over all components u_i .

$$u_i(Skew)_{u_i} = \sqrt{\rho} u_i \frac{\partial(\sqrt{\rho} u_i)}{\partial t} + \frac{1}{2} \left(\rho u_j u_i \frac{\partial u_i}{\partial x_j} + u_i \frac{\partial(\rho u_i u_j)}{\partial x_j} \right) \quad (2.5)$$

The expression for kinetic energy transport (2.5) can be rewritten as

$$u_i(Skew)_{u_i} = \frac{\partial(\frac{1}{2}\rho u_i^2)}{\partial t} + \frac{\partial(\frac{1}{2}\rho u_j u_i^2)}{\partial x_j}. \quad (2.6)$$

Multiplying the i^{th} momentum equation by u_i , and utilizing (2.6), we recover

$$\frac{\partial(\frac{1}{2}\rho u_i^2)}{\partial t} + \frac{\partial(\frac{1}{2}\rho u_j u_i^2)}{\partial x_j} + u_i \frac{\partial p}{\partial x_i} = u_i \frac{\partial \tau_{ij}}{\partial x_j}. \quad (2.7)$$

The convective term in the total energy equation is written using the divergence form $(Div.)_E$ to mimic the physical property that the total amount of energy in a closed system is conserved. Then, using total energy equation in (2.1) and subtracting the equation for kinetic energy (2.7), the expression for internal energy is recovered.

$$(Div.)_{e_{int}} + p \frac{\partial u_i}{\partial x_i} = \tau_{ij} \frac{\partial u_i}{\partial x_j} - \frac{\partial q_j}{\partial x_j} \quad (2.8)$$

2.1. Finite Difference Discretization

The method presented is a collocated Cartesian-grid scheme on a uniform mesh, with spacing Δx . All state variables and velocities are located at cell centers and occupy the same temporal level. Let the cell center be defined as $x_i = (i + 1/2)\Delta x$ and the cell face be defined as $x_{i-1/2} = i\Delta x$. The temporal level is denoted as $t^n = \sum_i \Delta t^i$ for a potentially non-uniform time step Δt^i at step i .

2.2. Discrete Operators

For the sake of completeness, the individual interpolation and differencing operators are presented. The Einstein summation convention is used for indicial subscripts, but note that repeated indices in interpolation superscripts are not summed. The spatial discretization is chosen such that the discrete scheme mimics the energy conserving properties of the continuous equations. The following interpolation and differencing

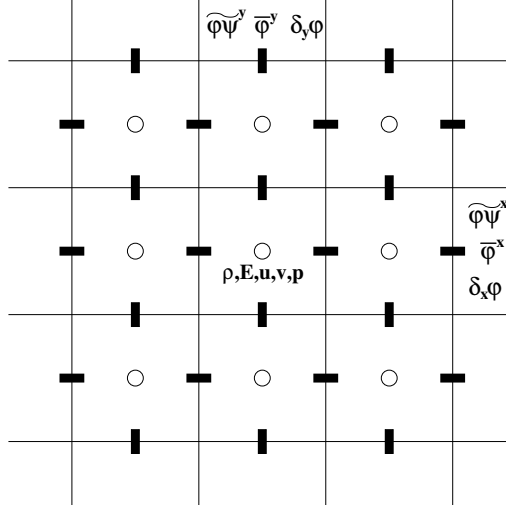


FIGURE 1. Locations of variables on the grid. Flow variables density, energy, pressure, and velocity are located at cell centers. For cell-centered variables φ and ψ , the quantities $\overline{\varphi}^x$, $\widetilde{\varphi}^x$, and $\delta_x(\varphi)$ are located on the east/west faces. Similarly, $\overline{\varphi}^y$, $\widetilde{\varphi}^y$, and $\delta_y(\varphi)$ are located on the north/south faces.

operators calculate a value interpolated from cell centers to cell faces or calculate a gradient on cell faces using cell centered values.

$$\overline{\varphi}^{x_1} \Big|_{x_{i+1/2,j}} = \frac{\varphi_{i+1,j} + \varphi_{i,j}}{2} \quad (2.9)$$

$$\delta_{x_1}(\varphi) \Big|_{x_{i+1/2,j}} = \frac{\varphi_{i+1,j} - \varphi_{i,j}}{\Delta x} \quad (2.10)$$

$$\widetilde{\varphi\psi}^{x_1} \Big|_{x_{i+1/2,j}} = \frac{\varphi_{i+1,j}\psi_{i,j} + \varphi_{i,j}\psi_{i+1,j}}{2} \quad (2.11)$$

When interpolating or differencing a quantity φ from cell centers to cell faces, (2.9-2.10) are second order accurate. Additionally, operator (2.11) interpolates the product $\varphi\psi$ to cell faces with second order accuracy as well. This can be shown by performing Taylor series analysis and expanding terms about the cell face of interest. The following discrete relations are used to demonstrate the various conservation properties of the method.

$$\delta_{x_1}(\overline{\varphi}^{x_j}) = \overline{\delta_{x_j}(\varphi)}^{x_j} = \frac{\varphi_{i+1,j} - \varphi_{i-1,j}}{2\Delta x} \quad (2.12)$$

$$\psi\delta_{x_j}(\overline{\varphi}^{x_j}) + \varphi\delta_{x_j}(\widetilde{\psi}^{x_j}) = \delta_{x_j}(\widetilde{\psi\varphi}^{x_j}) \quad (2.13)$$

$$\varphi \left(\delta_{x_j}(\psi\overline{\varphi}^{x_j}) + \overline{\psi\delta_{x_j}(\varphi)}^{x_j} \right) = \delta_{x_j}(\psi\widetilde{\varphi\varphi}^{x_j}) \quad (2.14)$$

Note that relation (2.14) assumes that ψ is a face-centered quantity. This property will be used in the discrete formulation of kinetic energy.

2.3. Semi-Discrete Second Order Finite Difference Formulation

The Navier-Stokes equations (2.1) are discretized such that mass and total energy are discretely conserved to machine precision on a domain with bi-periodic boundary

conditions. Define the face-centered momentum as

$$g_j = \overline{\rho}^{x_j} \overline{u_j}^{x_j}. \quad (2.15)$$

Then, the Navier-Stokes equations are discretized as

$$\frac{\partial \rho}{\partial t} + \delta_{x_j}(g_j) = 0 \quad (2.16)$$

$$\sqrt{\rho} \frac{\partial(\sqrt{\rho} u_i)}{\partial t} + \frac{1}{2} \left(\overline{g_j \delta_{x_j} u_i}^{x_j} + \delta_{x_j}(g_j \overline{u_i}^{x_j}) \right) + \overline{\delta_{x_i}(p)}^{x_i} = \overline{\delta_{x_j}(\tau_{ij})}^{x_j}. \quad (2.17)$$

$$\frac{\partial(\rho E)}{\partial t} + \delta_{x_j} \left(g_j \widehat{E}^{x_j} \right) + \delta_{x_j}(\widetilde{p u_i}^{x_j}) = \delta_{x_j}(\widetilde{\tau_{ij} u_i}^{x_j}) + \delta_{x_j}(\kappa \delta_{x_j}(T)) \quad (2.18)$$

The discrete total energy equation (2.18) uses an advective flux interpolation operator for the total energy. The form of the energy interpolation operator $\widehat{E}$ can be determined using the discrete kinetic energy equation.

$$\frac{\partial(\frac{1}{2} \rho u_i u_i)}{\partial t} + \delta_{x_j} \left(\frac{1}{2} g_j \widetilde{u_i u_i}^{x_j} \right) + u_i \overline{\delta_{x_i}(p)}^{x_i} = u_i \overline{\delta_{x_j}(\tau_{ij})}^{x_j} \quad (2.19)$$

and discrete internal energy equation

$$\frac{\partial(\rho e_{int})}{\partial t} + \delta_{x_j} \left(g_j \overline{e_{int}}^{x_j} \right) + p \overline{\delta_{x_j}(u_i)}^{x_i} = \tau_{ij} \overline{\delta_{x_j}(u_i)}^{x_j} + \delta_{x_j}(\kappa \delta_{x_j}(T)). \quad (2.20)$$

By requiring that the discretization of $(Div.)_E$ takes the form of

$$(Div.)_E = \delta_{x_j} \left(g_j \left(\overline{e_{int}}^{x_j} + \frac{1}{2} \widetilde{u_j u_j}^{x_j} \right) \right), \quad (2.21)$$

and using the definition of total energy (2.3), one finds that

$$\widehat{E}^{x_j} = \overline{E}^{x_j} - \frac{\Delta x^2}{4} \sum_i \delta_{x_j}(u_i)^2. \quad (2.22)$$

In 3D, this expression is expanded as

$$\widehat{E}^{x_j} = \overline{E}^{x_j} - \frac{\Delta x^2}{4} \left(\left(\frac{u_{j+1} - u_j}{\Delta x} \right)^2 + \left(\frac{v_{j+1} - v_j}{\Delta x} \right)^2 + \left(\frac{w_{j+1} - w_j}{\Delta x} \right)^2 \right). \quad (2.23)$$

The proper interpolation in 2D can be obtained by omitting the “ w ” terms from (2.23). Note that the discrete treatment of the energy and momentum terms necessitates use of specific energy.

The collocated tensor term is discretized in a novel fashion to remain consistent between the momentum and energy equations.

$$\tau_{ij} = \mu \left(\overline{\delta_{x_j}(u_i)}^{x_j} + \overline{\delta_{x_i}(u_j)}^{x_i} \right) - \frac{2}{3} \mu \overline{\delta_{x_k}(u_k)}^{x_k} \delta_{ij} \quad (2.24)$$

The momentum diffusion term that appears in (2.17) will have a stencil that exhibits “red-black” decoupling. However, this stencil satisfies the discrete product rule so that the discrete total energy equation can be written as the sum of the discrete internal and kinetic energy equations, as in Morinishi (2010). The standard 5-point 2D momentum diffusion stencil does not satisfy this property. See Appendix A for more details. This discretization is simultaneously consistent with the total, internal, and kinetic energy

equations. This ensures temporal convergence of both of the derived quantities, while exactly conserving total energy. See Sec. 8.2.

While the discretization of the transport, momentum/viscous dissipation, and force/work terms in the momentum and energy equations must be chosen to satisfy a discrete product rule, there is freedom to choose the discretization of the thermal diffusion term. As such, the standard 5-point stencil for the diffusion operator in 2D is used. This introduces coupling between adjacent cells which can mitigate the effects of using a centered-difference scheme for advection.

2.4. Turbulence Model

The above formulation is capable of performing high-fidelity DNS of turbulent flows, as we demonstrate in the Results Section of this report. Both 3D Taylor-Green vortex based turbulent and isotropic turbulent flows (2D and 3D) are considered later with. Moreover, we implement the standard Smagorinsky model for as an eddy viscosity model for very high-Reynolds number flows with moderate sized grids. Results from initial testing of this LES formulation appears promising.

3. Dynamic Block-Structured Adaptive Mesh Formulation

A block-structured adaptive grid method functions as a hierarchy of structured Cartesian grids on which computations are performed, where the domain of each fine mesh is a subset of the domain of every coarser mesh (Baden 2000; Berger & Colella 1989; Berger & LeVeque 1998b; Crutchfield & Welcome 1993). The grids on each level are divided into rectangular subdomains, or blocks. Block-structured methods allow for straight-forward implementation of numerical methods, as each block can be treated as its own domain. Boundary conditions of a block are typically implemented through a halo of surrounding “ghost cells.” A number of packages exist that implement the data structures and synchronization procedures for block-structured adaptive mesh refinement (AMR), facilitating researchers to use AMR without developing a large code base. Researchers can use the AMR framework to manage data structures, parallelism, and synchronization, while writing their own procedures to simulate the physics of interest. Such AMR frameworks include (but are not limited to) AMRClaw (Berger & LeVeque 1998a), AMROC (Deiterding 2003), BoxLib (Bell *et al.* 2012), Chombo (Colella *et al.* 2000), PARAMESH (MacNeice *et al.* 2000), and SAMRAI (Wissink *et al.* 2001).

The present work utilizes BoxLib (developed at Lawrence Berkeley National Laboratory (Bell *et al.* 2012)) for basic grid-based data structures on a single grid level and in the presence of multiple levels of mesh refinement. BoxLib provides a framework for performing computations on block-structured grids with dynamic AMR. Each grid level $\Omega^1 \supseteq \dots \supseteq \Omega^N$ is the union of a number of disjoint rectangular subgrids (hence, block-structured). The mesh resolution Δx^i on the i^{th} grid Ω^i is uniform in all directions and has the property that $\Delta x^{i+1} = \Delta x^i/2$.

In the presence of mesh refinement, coarse grid cells underlie each finer mesh, but are not active in the computation. The solution in covered coarse cells is overwritten in conservative fashion by the solution on the fine grid. As such, the integral of a conserved quantity on the fine grid is equal to the integral of the conserved quantity on the region of the coarse grid covered by the fine grid. At the boundary of coarse and fine grid cells, a buffer region of fine “ghost cells” is used to enforce continuity of the solution across the interface, as shown in Fig. 2. A fourth-order accurate, conservative method is used to fill the ghost cells, so that all discrete interpolation and differential operators are formally second-order accurate for differentiable solutions, even in the presence of mesh

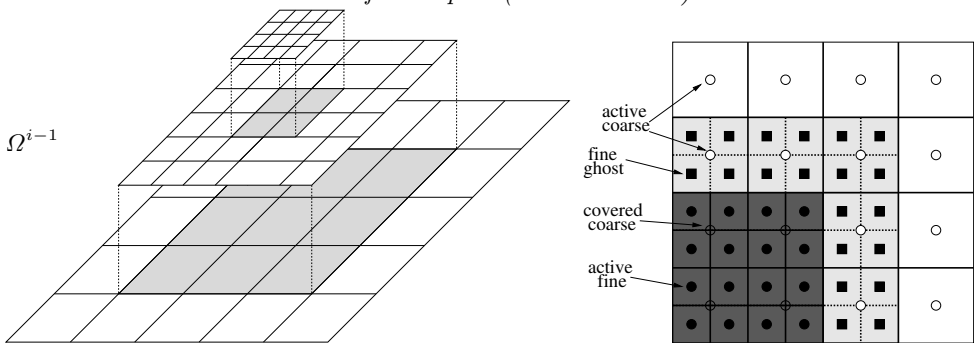


FIGURE 2. Left: Illustration of a properly nested mesh hierarchy. Right: Cells in a structured grid with mesh refinement, with coarse cells (○), fine cells (●), and ghost cells (■). The active coarse grid is shown in white; the active fine grid is has dark shading. The lightly shaded region is covered by fine “ghost cells,” used to enforce boundary conditions where coarse and fine meshes meet. The coarse cells covered by fine ghost cells are also active.

refinement. No slope-limiting is used, as the scope of this work is limited to shock-free, compressible flow.

Computations are performed successively on each grid level Ω^i , with computations being completed on each rectangular subgrid on that level. After all computation on a given level has been completed, the solution in any coarse grid cells covered by a fine mesh is overwritten in conservative fashion by the solution on the next finer grid level. Because computations are performed separately on each disjoint, rectangular subgrid, the method is naturally parallelizable. Methods using the BoxLib framework have been implemented and verified in a number of massively parallel applications. A discussion of the parallel performance of methods that use the Boxlib framework, such as CASTRO, MAESTRO, and NYX, can be found in [Almgren *et al.* \(2010a, 2013, 2007\)](#); [Zhang *et al.* \(2011a, 2013\)](#). It should be noted that while this method is implemented in BoxLib, it could be implemented in any block-structured framework, or on a single grid level.

3.1. Thermodynamic Variable Choice

There is some freedom in the formulation of the compressible Navier–Stokes equations (2.1), namely in the choice of the second state variable that is simulated in addition to density. [Wall *et al.* \(2002\)](#) select density and enthalpy in simulating low-Mach number compressible flow, while [Desjardins *et al.* \(2008\)](#) utilizes density and pressure obtained from the pressure Poisson equation. The works of [Morinishi \(2010\)](#); [Morinishi & Koga \(2014\)](#) advocate for employing density and internal energy. While the convective terms in the internal energy equation of [Morinishi \(2010\)](#); [Morinishi & Koga \(2014\)](#) are discretized in conservative fashion, the pressure dilatation and viscous heating terms appear in non-conservative form. This formulation of Navier–Stokes is observed to have positive attributes in total energy conservation on Cartesian meshes and in a deforming mesh framework ([Morinishi & Koga 2014](#)). However, this was found to have detrimental effects when using an adaptive grid hierarchy, including severe spurious oscillations at coarse/fine mesh boundaries. Because a non-dissipative advective scheme is used, these oscillations propagate and can potentially cause failure of the method.

It was found that use of the internal energy equation (2.20) in the context of AMR leads to the spurious concentration of energy and more severe pressure oscillations at coarse/fine mesh boundaries. Patch-based refinement methods are well-suited to equations that can be expressed in conservation form, as “re-fluxing” ([Almgren *et al.*](#)

1998; Berger 1987; Berger & LeVeque 1998b; Zhang *et al.* 2012) is used to ensure that fluxes are continuous at coarse/fine mesh boundaries. The internal energy equation (2.20) contains terms in non-conservation form, to which re-fluxing cannot be applied. It appears that C^1 errors at mesh boundaries leads to the spurious oscillations mentioned earlier. As such, the conservative total energy equation (2.18) is used, which conserves total energy to machine precision within the adaptive mesh hierarchy. This was found to decrease the severity of errors at coarse/fine mesh boundaries.

3.2. Staggered vs. Collocated AMR Methods

While a number of staggered grid methods for variable density, low-Mach number flow have been developed (Desjardins *et al.* 2008; Morinishi 2010; Wall *et al.* 2002), a staggered grid method was found to be not well-suited for use with the present work, which uses centered differencing methods in the context of adaptive mesh refinement. To retain formal second-order spatial accuracy of spatial operators at coarse/fine mesh boundaries for a staggered grid scheme, it is necessary to use interpolation stencils of higher order for both cell-centered and face-centered variables. A variety of fourth-order (and higher) interpolation methods exist for cell-centered variables (Ferm & Lötstedt 2004; McCorquodale & Colella 2011; Zhang 2011), and these methods can be used on variables located at cell centers for a staggered grid formulation. However, the irregular placement of face-centered variables in an multilevel grid hierarchy leads to the construction of operators with prohibitively large stencils, or one-sided stencils which are highly dissipative at coarse/fine mesh boundaries. These properties were found to be undesirable for the physical conditions being simulated in this work, leading to the selection of a collocated grid method for the present effort.

4. Immersed Boundary Formulation

A solid body is specified by altering the state variables that lie in the region occupied by the solid, such that certain boundary conditions are satisfied at the fluid/structure boundary. This methods are known as embedded geometry or embedded boundary methods. Such a method is implemented so that the Navier-Stokes equations can be applied uniformly throughout the domain, without the need to recompute the fluid stencil due to displacement of the solid. The location of the interface is specified by a level-set function, a signed distance function that is positive in the fluid, negative in the solid, and zero on the interface (Peng *et al.* 1999; Sethian & Smereka 2003; Smereka 2006). A front-propagation technique based on the works of (Chen *et al.* 1997; Hou *et al.* 1997; Osher & Sethian 1988; Uddin *et al.* 2014; Wu *et al.* 2007) is used to implement the embedded boundary in the flow.

Suppose some interface Γ sub-divides $\mathbb{R}^n$ such that

$$\mathbb{R}^n = \Omega_1 \cup \Omega_2, \quad \Omega_1 \cap \Omega_2 = \emptyset. \quad (4.1)$$

Then, level-set function ϕ associated with the interface Γ is a function

$$\phi : \mathbb{R}^n \rightarrow d \subseteq \mathbb{R}, \quad (4.2)$$

where the domain is the set of points in physical space, and the range is the shortest distance from a point to the interface Γ . The sign of the level-set function can be defined

such that

$$\phi(\mathbf{x}) = \begin{cases} > 0, & \mathbf{x} \in \Omega_1 \\ < 0, & \mathbf{x} \in \Omega_2 \\ = 0, & \mathbf{x} \in \Gamma. \end{cases} \quad (4.3)$$

If an explicit level-set function is unavailable, an approximate level-set function can be calculated by computing the orthogonal projection of the argument point onto the interface and computing the distance from the argument to the orthogonal projection.

4.1. Characteristic Propagation Method

Similarly to the work of Uddin *et al.* (2014), we will assume that the extrapolation of the fluid field $f(\mathbf{x})$ into the solid can be approximated by the multi-dimensional Taylor expansion about interfacial point $\mathbf{x}_\Gamma$

$$f = f_0 + \nabla f \cdot (\mathbf{x} - \mathbf{x}_\Gamma) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_\Gamma) \cdot H(\mathbf{x} - \mathbf{x}_\Gamma). \quad (4.4)$$

Additionally, we will assume that for some functions $\beta_0(\mathbf{x})$, $\beta_1(\mathbf{x})$, and $\beta_2(\mathbf{x})$, we can parametrize the function f in terms of the value of the level-set function (4.5). Recall that the level-set function ϕ is a measure of normal distance to the interface, so (4.5) can be understood as a parametrization of the fluid field in terms of normal distance.

$$f(\mathbf{x}) \approx \beta_0(\mathbf{x}) + \beta_1(\mathbf{x})\phi(\mathbf{x}) + \beta_2(\mathbf{x})\phi^2(\mathbf{x}), \quad (4.5)$$

Write an arbitrary point $\mathbf{x}$ in the fluid as $\mathbf{x} = \mathbf{x}_\Gamma + \hat{n}\phi$, for some interfacial point $\mathbf{x}_\Gamma$ with outward unit normal $\hat{n}$. Then by substituting the expression for $\mathbf{x}$ into (4.4) and matching terms with (4.5), we find that on the interface,

$$\begin{aligned} \beta_0 &= f_0 \\ \beta_1 &= \nabla f \cdot \hat{n} \\ \beta_2 &= \frac{1}{2}\hat{n} \cdot H\hat{n}. \end{aligned} \quad (4.6)$$

Note that (4.6) are the values taken by the functions $\beta_0(\mathbf{x})$, $\beta_1(\mathbf{x})$, and $\beta_2(\mathbf{x})$ on the boundary (i.e. the fluid/structure interface). To find the values of these functions in the solid, and thus reconstruct the field f , we must solve a differential equation for each β_i . Two important properties of a level-set function ϕ will be used:

(i) The gradient of the level-set function has magnitude one ($|\nabla\phi| = 1$) for all points such that the closest orthogonal projection onto the surface is unique.

(ii) The gradient of the level-set function on the fluid/solid interface is normal to the interface, $\nabla\phi = \hat{n}$.

The PDE (4.7) is solved inside the solid structure to extrapolate values from the fluid into solid in conjunction with appropriate boundary conditions.

$$\begin{cases} \text{sgn}(\phi) (\nabla\phi \cdot \nabla\beta_i) = 0 \\ \beta|_\Gamma = f(\mathbf{x}) \end{cases} \quad (4.7)$$

This PDE can be interpreted as the steady-state of advection equation for β_i in pseudo-time τ , in which information propagates from the boundary into the solid with a velocity of unit magnitude that is normal to the interface. In this context, the level-set function is negative in the solid, so the advective velocity can be interpreted as

$$-\nabla\phi = -\hat{n}.$$

It should be understood that pseudo-time τ is not related to physical time t in the Navier-Stokes equation (2.1). At each stage in the discrete equations, when the solution changes in the fluid region, the extrapolation PDE (4.7) should be solved so that the appropriate boundary conditions are enforced at the fluid/structure interface. Due to the second order discretization of the Navier-Stokes equations, the stencil requires at most 2 cells inside the solid (the $\frac{\partial \tau_{ij}}{\partial x_j} x_j$ term has a bandwidth of 5 cells in each dimension).

To calculate the necessary boundary conditions (4.6), a least squares procedure is used, as in Uddin *et al.* (2014), to approximate fluid quantities on the interface Γ . Values in the fluid are collected in a small neighborhood about a cut cell, where a cell is said to be “cut” if $\phi(\mathbf{x}_{i,j}) < 0$ and some neighbor $\phi(\mathbf{x}_{i',j'}) \geq 0$, with $i', j' = -1, 0, +1$. The entries of the least-squares reconstruction matrix are then

$$A = \begin{bmatrix} 1 & \vec{\delta x} & \vec{\delta y} & \frac{1}{2} \vec{\delta x}^2 & \delta x \vec{\delta y} & \frac{1}{2} \vec{\delta y}^2 \end{bmatrix}, \quad (4.8)$$

where the neighboring cells of the cut cell are ordered in a vector. The system

$$A\vec{c} = \vec{r} \quad (4.9)$$

is solved in the least squares sense, where

$$\vec{r} = [f_1 \quad f_2 \quad \dots \quad f_N]^T \quad (4.10)$$

$$\vec{\delta x} = [(x_1 - x_\Gamma) \quad (x_2 - x_\Gamma) \quad \dots \quad (x_N - x_\Gamma)]^T \quad (4.11)$$

$$\vec{\delta y} = [(y_1 - y_\Gamma) \quad (y_2 - y_\Gamma) \quad \dots \quad (y_N - y_\Gamma)]^T \quad (4.12)$$

and the solution vector is

$$\vec{c} = \begin{bmatrix} f_0 & \frac{\partial f}{\partial x} & \frac{\partial f}{\partial y} & \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}_\Gamma^T. \quad (4.13)$$

Here, let $\mathbf{x}_i$ denote a point in the fluid (i.e. $\phi(\mathbf{x}_i) > 0$) used to calculate the least-squares reconstruction at $\mathbf{x}_\Gamma$. The values in the matrix A are the ordered offsets appearing in (4.4). The values f_i appearing in $\vec{r}$ represent the flow variable value in the fluid at point $\mathbf{x}_i$. Once the linear system (4.9) has been solved in the least-squares sense, β_0, β_1 , and β_2 can be computed. Then, the extrapolation differential equation (4.7) is solved for $\beta_i(\mathbf{x})$ inside the solid. After solving (4.7), the reconstruction of the flow variable in the solid is calculated as (4.5).

It should be noted that (4.7) is recovered in the limit as $\epsilon \rightarrow 0$ in the work of Uddin *et al.* (2014), which includes a diffusive term (4.14).

$$\text{sgn}(\phi) (\nabla \phi \cdot \nabla \beta_i) = \epsilon \Delta x \nabla \cdot \left(\left| \frac{\phi}{\Delta x} \right|^{i+1} \nabla \beta_i \right) \quad (4.14)$$

This diffusive term was included to prevent the formation of sharp gradients in the interior of the solid, where characteristic paths emanating from the boundary intersect. The method in Uddin *et al.* (2014) was posed for use with discretizations with wide (or even global stencils). For this work, however, it was found the the accuracy of the flow simulation was largely insensitive to even drastic changes in the diffusive parameter (testing $\epsilon \in [10^{-10}, 10^4]$), as the discretized Navier-Stokes equations require at most only two cells in the solid. Additionally, the PDE (4.14) becomes increasingly stiff as the solid is better resolved, if we require convergence of the extrapolation within the entire

solid body. The value $\left| \frac{\phi}{\Delta x} \right|^{i+1}$ acts as a diffusive parameter, where $\frac{\phi}{\Delta x}$ is approximately the number of cell widths from a given point to the interface. A well-resolved solid (e.g. a circle with $R/\Delta x > 15$) requires aggressive preconditioning for a robust iterative procedure such as the Biconjugate Gradient Stabilized (BiCG-Stab) method to converge for all cells within the solid. As such, the PDE (4.7) is solved using an iterative Gauss-Seidel procedure that iterates a uniformly zero initial condition in the solid to satisfactory convergence for cells such that $|\phi(x_{i,j})| < 3\Delta x$. Because convergence is required only in a band near the interface, rather than “globally” within the solid, the method scales well with mesh resolution. Use of an iterative “marching” scheme lends itself naturally to multilevel grid scheme.

4.1.1. Characteristic Propagation Method Discretization

The discretization of (4.7) is calculated dynamically, using the level-set function ϕ at the current time step to determine the location of the interface and thus the altered length fraction that appears in the difference stencil. The discretization of (4.7) will be presented in the x-direction, but the discretization is identical in each cardinal direction.

Let Γ^n be the location of the interface at time t^n , and let ϕ^n be the corresponding level-set function. Then, we will define the advective velocity a as

$$a \equiv \text{sgn}(\phi) \frac{\partial \phi}{\partial x}. \quad (4.15)$$

Additionally, we will define the sign of a as $s \equiv \text{sgn}(a)$. To define s , $\frac{\partial \phi}{\partial x}$ is calculated using a centered difference, rather than upwinding. Then, the level-set function is approximated as

$$\phi(x) = \phi(x_i) + (x - x_i) \frac{\partial \phi}{\partial x}. \quad (4.16)$$

On the interface, $\phi = 0$, so the length fraction ℓ occupied by the solid can be calculated as

$$\ell = -\frac{\phi(x_i)}{\Delta x \frac{\partial \phi}{\partial x}}. \quad (4.17)$$

Assume that the interface has an intercept in the range $[x_{i-1}, x_{i+1}]$. The derivative is approximated as

$$\frac{\partial \phi}{\partial x} \approx s \frac{\phi_i - \phi_{i-s}}{\Delta x}, \quad (4.18)$$

and the length fraction written as

$$\ell = \frac{s\phi_i}{\phi_{i-s} - \phi_i} \quad (4.19)$$

Defining $a^+ = \max(a, 0)$ and $a^- = \min(a, 0)$, we are able to write the discretization of (4.7) in a general form, accounting for the cell being cut arbitrarily on the interval $[x_{i-1}, x_i]$ or $[x_i, x_{i+1}]$. For an uncut cell x_i and an arbitrary field β ,

$$a \frac{\partial \beta}{\partial x} \approx a^+ \frac{\beta_i - \beta_{i-1}}{\Delta x} + a^- \frac{\beta_{i+1} - \beta_i}{\Delta x} = |a| \frac{\beta_i - \beta_{i-s}}{\Delta x}. \quad (4.20)$$

For a singly-cut cell, with value β_{Γ^x} on the interface, the advective derivative is approximated as

$$a \frac{\partial \beta}{\partial x} \approx a s \frac{\beta_i - \beta_{\Gamma^x}}{|\ell| \Delta x} = |a| \frac{\beta_i - \beta_{\Gamma^x}}{|\ell| \Delta x}. \quad (4.21)$$

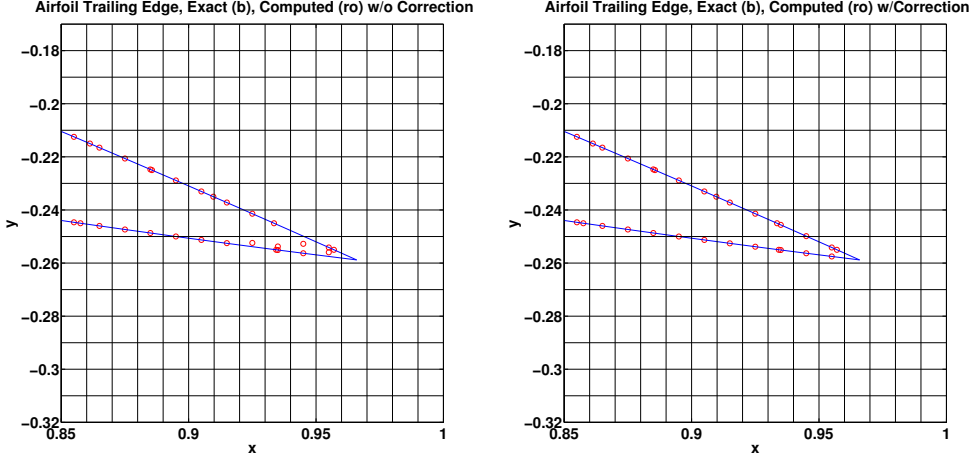


FIGURE 3. Results of the level-set root-finding procedure on the trailing edge of a NACA-0012 airfoil with a non-zero angle of attack. Results are shown without (left) and with (right) the correction (4.22), with the exact interface in blue, and the computed interface shown as red circles. Note the errors in the under-resolved region when the correction is not used (left).

4.1.2. Doubly-Cut Cells

It must be noted that it is possible for a cell to be cut by the interface between both $[x_{i-1}, x_i]$ and $[x_i, x_{i+1}]$. In this case, the features of the structure is under-resolved in cell x_i . Such doubly-cut cells can occur at the trailing edge of an airfoil, where the curves denoting the top and bottom edges meet at a single point. This case requires special care, as (4.19) will fail to accurately compute the location. This is due to the fact that the level-set function computes the *shortest* distance from a point to the interface. So, the level-set function will take the value of the distance to either the top edge or bottom edge (whichever is closest), but not both simultaneously.

Let Γ_1 and Γ_2 denote the top and bottom edge of an airfoil, and suppose that $|\phi_{\Gamma_1}(x_i)| < |\phi_{\Gamma_2}(x_i)|$. The ϕ_i will accurately predict the distance from x_i to Γ_1 while under-predicting the distance to Γ_2 . This under-prediction causes (4.19) to inaccurately calculate the location of the interface. A simple correction can be applied if a doubly-cut cell is detected by replacing (4.19) with

$$\ell^\pm = 1 - \frac{\phi_{i\pm 1}}{\phi_{i\pm 2} - \phi_{i\pm 1}}, \quad (4.22)$$

where the sign is taken as positive for the intercept in $[x_i, x_{i+1}]$ and negative for the intercept in $[x_{i-1}, x_i]$. The formula (4.22) can be recovered by writing a Taylor series expansion for $\phi(x)$ about $x_{i\pm 1}$ and requiring that the length fraction ℓ is positive. Note that calculation of neither intercept uses the value ϕ_i , as it is not known *a priori* which intercept is more accurately predicted by value ϕ_i . See Fig. 3.

Clearly, the discretization of (4.20) must be altered for cells cut on both sides, potentially in multiple directions. Rather than revert to a diffusion operator, we alter the PDE. Let us write the two-dimensional form of (4.20).

$$\mathbf{a} \cdot \nabla \beta \approx a_x^+ \frac{\beta_{i,j} - \beta_{i-1,j}}{\Delta x} + a_x^- \frac{\beta_{i+1,j} - \beta_{i,j}}{\Delta x} + a_y^+ \frac{\beta_{i,j} - \beta_{i,j-1}}{\Delta y} + a_y^- \frac{\beta_{i,j+1} - \beta_{i,j}}{\Delta y} \quad (4.23)$$

Now, suppose that the cell (i, j) is doubly-cut in the y-direction. Then, we assume a linear interpolant in the y-direction (with value β_* at $\beta_{i,j}$) and alter the stencil accordingly. We

will enforce the additional constraint that if the cell is cut on all four sides, the stencil should be non-degenerate. Note that for $\ell^+ = \ell^-$, the linear interpolant is second order accurate, and first order otherwise.

$$\mathbf{a} \cdot \nabla \beta \approx a_x^+ \frac{\beta_{i,j} - \beta_{i-1,j}}{\Delta x} + a_x^- \frac{\beta_{i+1,j} - \beta_{i,j}}{\Delta x} + \frac{1}{2} \beta_{i,j} = \frac{1}{2} \beta_{*,y} \quad (4.24)$$

This differs from Uddin *et al.* (2014), which uses an altered diffusion stencil in (4.14) to account for multiply-cut cells.

4.2. Adaptive Characteristic Propagation Method

Here, we introduce a novel technique for applying embedded geometry methods such as Uddin *et al.* (2014) in the context of Adaptive Mesh Refinement (AMR). This differs from other methods that use the IBM in conjunction with AMR, in that we allow the fluid/solid interface to be cut by levels of mesh refinement. Suppose that we introduce an n -level adaptive mesh hierarchy

$$\Omega^1 \supset \Omega^2 \supset \dots \supset \Omega^n,$$

where Ω^i is the region in $\mathbb{R}^n$ occupied by mesh level i with uniform resolution Δx_i , and $\Delta x_{i+1} < \Delta x_i$. The property that Ω^{i+1} is a proper subset of Ω^i is known as *proper nesting*, where a fine grid is contained in the union of all grids on the next coarsest mesh level. The concept of nesting is typical of patch-based AMR methods such as Almgren *et al.* (2010b); Berger & Colella (1989); Berger & LeVeque (1998b), as it simplifies possible mesh topologies. Additionally, it is required that a certain number of grid cells with resolution Δx_i are required between cells of resolution Δx_{i+1} and Δx_{i-1} . This ensures that all computational stencils can be properly filled.

Let us define Ω_S as the region occupied by the solid and Ω^r as the finest level such that

$$\Omega_S \subset \Omega^r \quad (4.25)$$

$$\Omega_S \cap (\Omega^i)^C \neq \emptyset, i > r. \quad (4.26)$$

Here, $(\Omega^i)^C$ is the standard complement operator. Additionally, let $\partial\Omega_S$ represent the boundary of the solid, i.e. the fluid/solid interface. Because the solid body is entirely contained by mesh Ω^r , the standard methods described in Uddin *et al.* (2014) can be applied. However, in the case of high Reynolds number flows, it may be necessary to use a higher level of mesh resolution than Δ_r to resolve certain aspects of the flow or regions of the solid body. If adaptive grids are generated based on flow characteristics or body features, levels Ω^i , for $i > r$, may intersect Ω_S in an arbitrary fashion. Rather than directly solve the PDE from Section 4.1 simultaneously on all levels, it is preferable to solve (4.7) to satisfactory convergence on each level sequentially, using the solution on the coarse mesh to define boundary conditions on the fine mesh. The following algorithm is used to solve (4.7) in the presence of an adaptive mesh hierarchy.

- (i) Set $i = r$.
- (ii) Solve (4.7) to convergence in pseudotime τ on Ω^i .
- (iii) If $i < n$
 - (a) Use the converged solution on Ω^i to fill coarse/fine ghost cells on level Ω^{i+1} .
 - (b) Compute the appropriate boundary conditions (4.6) on the interface $\partial\Omega_S \cap \Omega^{i+1}$.
 - i. If the least-squares stencil cannot be filled on Ω^{i+1} , calculate the boundary conditions using data from level Ω^i .



FIGURE 4. Modified least-squares stencil at a coarse/fine mesh boundary. Showing the fine grid Lagrangian interface point (open circle), under-filled fine level least-squares stencil (closed square), and coarse level least-squares stencil (closed circle). In practice, a 5x5 or 7x7 stencil is used so that (4.9) is not rank deficient.

(c) Set $i = i + 1$

(d) Go to Step 2.

Coarse/fine boundary conditions need not be filled on level Ω^r , as the solid is entirely contained on mesh level r , i.e. $(\Omega^r)^C \cap \Omega_S = \emptyset$. In the case that the least-squares stencil cannot be filled on level Ω^i , the Lagrangian interface point is near a coarse/fine mesh boundary. It is assumed that accuracy on the coarse grid is acceptable in this region, as the mesh is unrefined on the “coarse” side of the coarse/fine boundary. So, the least-squares stencil is filled using data from Ω^{i-1} to calculate the boundary conditions. See Fig. 4. Integral quantities, such as lift and drag are computed on level Ω^r using the discrete Dirac delta function derived in Smereka (2006). refinement present at the interface.

5. Solid Mechanics Model Development

5.1. Viscoelastic and Electrostrictive Constitutive Modeling

In the following discussion, the hyperelastic, viscoelastic and electrostrictive governing equations are summarized. Internal variables associated with rate dependent dissipation are developed following the general thermodynamic framework outlined in Gurtin (1967); Holzapfel & Simo (1996); Peng *et al.* (1977). The origins of the dissipative equation governing internal loss based on the second law of thermodynamics is developed to provide guidance to the integration of the hyperelastic energy functions and coupling with viscoelastic dissipative energy functions. This provides a more general framework than elastic models which may be directly integrated into complex damping in helicopter blade studies, for example. Through this approach, we develop linear and nonlinear forms of the viscoelastic stress and analyze its uncertainty using Bayesian statistics.

The following thermodynamic framework includes thermal effects, but will focus on model validation of isothermal deformation processes. The total free energy density function including non-conserved order parameters is defined by

$$\psi = \psi_\infty(F_{iK}, \Theta, \tilde{D}_K) + \Upsilon(F_{iK}, \Theta, \Gamma_{iK}^\alpha) \quad (5.1)$$

per reference volume where ψ_∞ is the conserved, hyperelastic free energy function and

Υ is an energy function that contains non-conserved internal states denoted by Γ_{iK}^α . The hyperelastic free energy is a function of F_{iK} (the deformation gradient (Malvern 1969)), temperature Θ , and electric displacement $\tilde{D}_K$ (for electromechanically coupled problems), while Υ includes Γ_{iK}^α which are a set of internal variables ($\alpha = 1, \dots, n$ non-measurable internal states) that contribute to dissipation during rate dependent deformation.

The nominal stress can be directly obtained from these energy functions as

$$s_{iK} = \frac{\partial \psi}{\partial F_{iK}} = \frac{\partial \psi_\infty}{\partial F_{iK}} + \frac{\partial \Upsilon}{\partial F_{iK}}. \quad (5.2)$$

The additional relations associated with viscoelastic stresses and evolution of the internal state are neglected for brevity. Most importantly, is the second law equation which must be satisfied when determining stress due to losses with a hyperelastic or elastic structure. This requires

$$-\frac{\partial \psi}{\partial \Gamma_{iK}^\alpha} \dot{\Gamma}_{iK}^\alpha - \frac{Q_I \Theta_{,I}}{\Theta^2} \geq 0 \quad (5.3)$$

where the second term is associated with heat flux. These two terms describe the entropy production. In cases where the thermal gradients are negligible but viscoelastic effects are present, the first term on the right hand side must be positive definite. Following Peng *et al.* (1977), we assume entropy production is a function of the time rate of change of the internal state variable and the deformation gradient

$$-\frac{\partial \hat{\psi}}{\partial \Gamma_{iK}^\alpha} \dot{\Gamma}_{iK}^\alpha = Q_{iK}^\alpha \dot{\Gamma}_{iK}^\alpha = F(\dot{\Gamma}_{iK}^\alpha, F_{iK}) \geq 0 \quad (5.4)$$

where both the viscoelastic stress and the entropy production function $F(\dot{\Gamma}_{iK}^\alpha, F_{iK})$ are unknown. We determine the appropriate form of the entropy generation function in combination with the hyperelastic function using Bayesian statistics.

5.2. Electrostriction of Membranes

When these constitutive relations are implemented within an adaptive structure, it becomes important to understand how to control various properties of the material in a fluid medium. For example, in certain thin plates, membranes, and shells, it is often desirable to apply a field through such structures in order to change the deformation state for flow control applications. An example of such a smart material in the form of a soft membrane is relevant to real-time control of lift and drag in low Reynolds number flows (Hays *et al.* 2013). To integrate such behavior into an adaptive fluid-structure system, one must understand the electrostrictive response of the material. The electrostrictive stress (σ_{jk}^e) is given as

$$\sigma_{jk}^e = \kappa \left(E_j E_k - \frac{1}{2} E_i E_i \delta_{jk} \right) \quad (5.5)$$

where E_j is the applied electric field in the j -direction, κ is the material's dielectric constant, and δ_{ij} is the Kronecker-delta function.

The structure used to take advantage of electrostriction consists of a membrane with a perforated center hole for loading the structure. A schematic of this structure is shown in Figure 5. The non-deformed configuration can be seen in Figure 6(a). Electrodes are placed on either side of the membrane to control the membrane with an electric field. Additionally, a transverse load can be applied to the center ring as seen in Figure 6b. This representation assumes a homogeneous profile through the radius of the membrane, but this is simply an approximation. Importantly, the membrane will

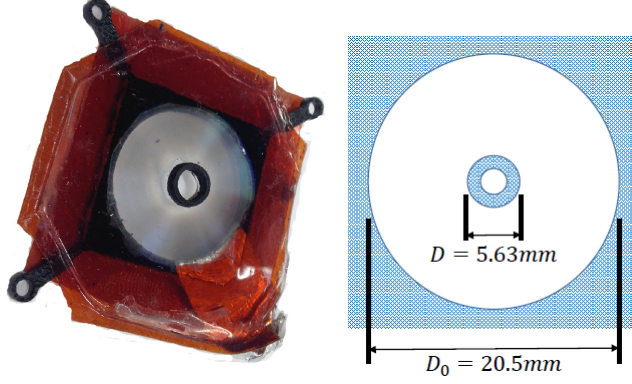


FIGURE 5. (a) Elastomer is stretched over acrylonitrile butadiene styrene (ABS) plastic frame and electrically isolated for electrode application. (b) Top view of experimental setup.

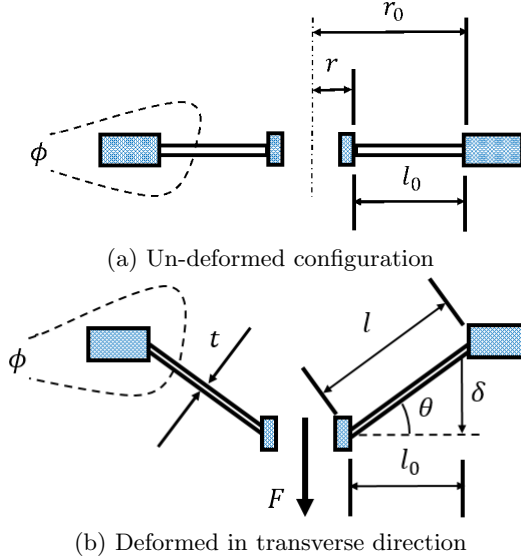


FIGURE 6. Cross section illustration of the membrane structure. The structure is loaded along the surface of the center ring. The load is denoted by F and is applied in the transverse direction. Inhomogeneous deformation along l is neglected in the reduced-order model.

deform inhomogeneously in a manner similar to that observed in Figure 7. In light of the complexities associated with this structure, the analysis of the loading is broken into several components.

Considerable research has been done on identifying the dielectric constant for these types of materials (Kofod *et al.* 2003; Wissler & Mazza 2007; Zhao *et al.* 2007). Interpreting the constitutive behavior in light of uncertainty is important and often neglected in materials and mechanics research. Testing of the material on the design platform reveals a non-trivial relationship between the material's dielectric properties and the deforming motion of the structure as has been summarized in detail elsewhere (Miles *et al.* 2015a). To highlight this complex relationship, the dielectric behavior of the adaptive structure was quantified and compared with deformation and electrostrictive coupling.

A variety of structural uncertainty was identified and validated. These effects can

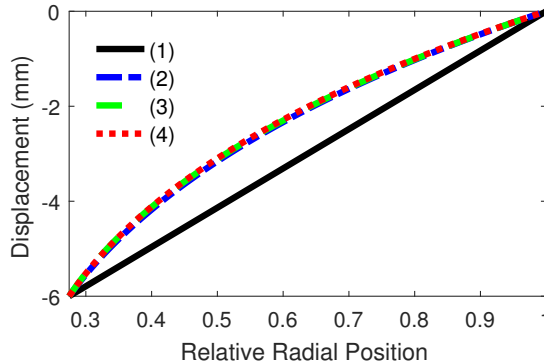


FIGURE 7. The deformation profile is different depending on the model being used. The lines represent: (1) homogeneous model, (2) inhomogeneous model with no applied field, (3) inhomogeneous model with 6 kV applied field, and (4) inhomogeneous model with 6 kV applied field and deformation dependent permittivity.

manifest from: 1) bi-axial pre-stretching, 2) transverse mechanical loading, and 3) application of electrostatic fields. Loading of this nature has been considered previously but in a spring loaded configuration (Rizzello *et al.* 2014), which has been used in a family of commercial devices known as universal muscle actuators UMA's, marketed by Artificial Muscle, Inc. In this model, the applied forces must be balanced with the nonlinear geometric effects of finite deformation, nonlinear constitutive relations.

A nonaffine hyperelastic constitutive model has yielded comparable results to more complex models (Miles *et al.* 2015b). The nonaffine model is advantageous because it reduces the number of model parameters and is less phenomenological. Important to the structural analysis, we consider both a homogeneous kinematic approximation and a higher fidelity inhomogeneous deformation profile as seen in Figure 7 (Zhao *et al.* 2007). While it is known that the true deformation profile is inhomogeneous, it is important to determine whether a simpler homogeneous model can still yield sufficiently accurate results for certain adaptive structure and fluid-structure applications.

Uncertainty analysis has been used in assessing the homogeneous deformation model, revealing the potential existence of deformation dependent permittivity within the material (Miles *et al.* 2015a). Bayesian methods worked well for identifying the unknown model parameters, but this study reveals the importance of testing calibrated model parameters with non-trained data. The calibration was performed by using data from one experiment (electric displacement-electric field (D-E) hysteresis cycles). The calibrated parameters were then used to predict the behavior of a completely different experiment (transverse deformation with electrostatic field), yielding results that did not agree well with data (see Figure 8). This discrepancy indicates the potential dependence that permittivity has with respect to deformation.

In order to confirm these results, an assessment of the inhomogeneous deformation model must be performed to determine if the simplifying assumptions within the homogeneous model created the appearance of deformation dependent permittivity. Analysis in light of uncertainty is non-trivial as the inhomogeneous model is not currently amenable to Bayesian methods. Simple optimization studies have revealed a similar dependence between the permittivity and transverse deformation, but further analysis beyond the scope of the current research is required to confirm this behavior. It may also prove necessary to consider the viscoelastic nature of the material in the context of determining the electrostrictive characteristics.

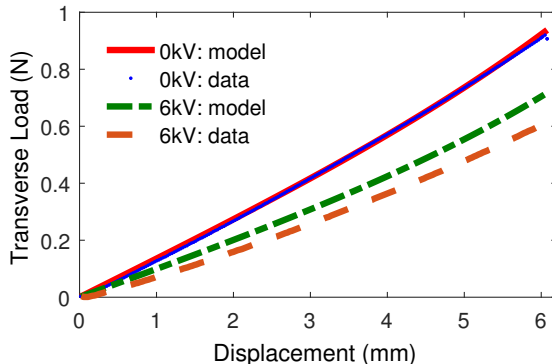


FIGURE 8. Using the relative permittivity found from the calibrating the electric displacement-electric field (DE) hysteresis cycles, we predict the effect of electrostriction during transverse loading. At zero field there is good model agreement as expected. Upon application of a larger field the model underpredicts the effects of electrostriction.

5.3. Bayesian Statistical Analysis

The Bayesian statistical analysis was implemented to quantify and analyze parameter uncertainty associated with the hyperelastic, viscoelastic, and electrostrictive parameters. This was conducted by comparing the model results to a series of rate dependent stress versus stretch data and field-driven mechanical measurement summarized elsewhere (Miles *et al.* 2015c)[†].

Underlying this parameter uncertainty method is Bayes' relation

$$\pi(\theta|y) = \frac{p(y|\theta)\pi_0(\theta)}{\int_{R_p} p(y|\theta)\pi_0(\theta)d\theta} \quad (5.6)$$

where $\pi(\theta|y)$ is the posterior probability of the observed data (y) given the viscoelastic model. $p(y|\theta)$ is the likelihood that incorporates information provided by the data, $\pi_0(\theta)$ is the prior probability, and the expression in the denominator is the normalization factor such that the integral of the posterior density over all possible parameter values is equal to one. A comparison of the error between the model and the data for a range of stretch rates is shown in Figure 9.

6. Structure Code Development

This problem involves the interaction of fluids with some type of solid structure. There are several forces at work acting on the structure. From the fluid, there is an external load from pressure, and there may also be additional loading from some applied voltage. The fluid modeling provides information on the pressure load which is subsequently used to calculate the force acting on the structural member. The equations of motion governing the behavior of the member are used to model the position and velocity, which is in turn sent back to the fluid's computations.

The structures modeling has been focused on developing two different Fortran based codes. The first model looks at the structure as an idealized beam, giving essentially a 1-D problem. The second model is more general and looks at the structure as a plate (2-D). Both models used cubic-spline basis functions. The system being solved for is given in Equation (6.1). There are variations to specific components of the left- and

[†] Received SMASIS best paper reference.

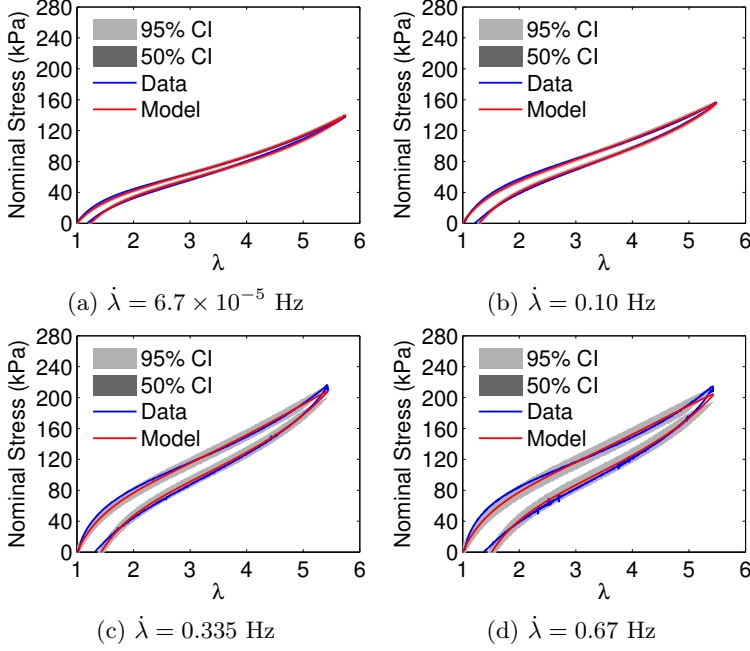


FIGURE 9. Hyperelastic, nonlinear viscoelastic model fits for a range of stretch rates. CI denotes the prediction and credible intervals based on the uncertainty in the measurements and model parameter uncertainty.

right-hand sides depending on which model is being used (beam, plate, etc.), but the basic representation is uniform.

$$\mathbb{M}\ddot{\mathbf{x}} + \mathbb{Q}\dot{\mathbf{x}} + \mathbb{K}\mathbf{x} = \mathbf{f} + V(t)\mathbf{b} \quad (6.1)$$

where $\mathbb{M}$, $\mathbb{Q}$, and $\mathbb{K}$ and the *global* mass, damping and stiffness matrices, respectively, and $\mathbf{f}$ and $\mathbf{b}$ are both forcing terms. The external loading due to pressure would be included in $\mathbf{f}$, whereas forces such as that caused by an electromagnetic field would be included in $\mathbf{b}$.

6.1. Dynamic Equation

The left- and right-hand sides of Equation (6.1) are both defined with respect to cubic B-Splined basis functions, which are discussed in [Smith \(2005\)](#). For the sake of simplicity, the global matrices are given in terms of the beam idealization model (1-D). The global mass matrix is defined in Equation (6.2).

$$[\mathbb{M}]_{jk} = \int_0^L \rho \phi_j \phi_k \, dx \quad (6.2)$$

where ρ is the material density ($\frac{kg}{m^3}$), ϕ_j and ϕ_k are the cubic spline basis functions. It is calculated across the entire length L . The global damping is defined as follows,

$$[\mathbb{Q}]_{ij} = \int_0^L [\gamma \phi_i \phi_j + cI \phi_i'' \phi_j''] \, dx \quad (6.3)$$

where γ is the external damping parameter that is caused by air and other external media ($\frac{Ns}{m^2}$), cI is the internal damping parameter ($\frac{Ns}{m^2}$), and ϕ_i'' and ϕ_j'' are the second derivatives of the B-Spline functions. The global stiffness matrix is very similar to the

second component of the damping matrix.

$$[\mathbb{K}]_{ij} = \int_0^L YI \phi_i'' \phi_j'' dx \quad (6.4)$$

where YI is the stiffness parameter and ϕ_i'' and ϕ_j'' are defined as before.

Lastly note that key coding development was applied to distribute the forcing term caused by the pressure load from the fluid. This term is defined by

$$[\mathbf{f}]_i = \int_{\Omega} f_n \Phi_i d\omega \quad (6.5)$$

where f_n is the normal acting force caused by pressure. In the case of electroactive coupling, we have the additional term relevant to adaptive structures

$$[\mathbf{b}]_i = \int_{\Omega} f \Phi_i' d\omega. \quad (6.6)$$

6.2. Approximate Solution

Each element is partitioned along its entire length L . Looking in the x -direction, with total length L , the length of each element (h_x) is equal to the total length divided by the number of grid points (N_x), i.e., ($h_x = \frac{L}{N_x}$). Now, x_p is equal to ph_x where $p = 0, \dots, N_x$; this would produce a vector indicating the location of nodes in the x -direction.

Cubic spline basis functions are used to map the information onto the structural member. We can relate the approximate solution for the beam model to the cubic B-Splines in the following way:

$$w^N(t, x) = \sum_{j=1}^{N+1} w_j(t) \phi_j(x) \quad (6.7)$$

where w_j is the displacement, and ϕ_j is the B-Spline function.

7. Fluid-Structure Interactions

Based on the structural mechanics solver, the position of the surface boundary is updated over time. The structure receives the information about the pressure and shear stress distribution along the surface from the fluid solver to determine the motion and deformation of the structure. This information is then in turn provided to the immersed boundary routine of the fluid flow solver to provide appropriate position and velocity of the solid surface. The time integration is then coupled in an iterative manner or in an predictor-corrector fashion. In the current formulation, we follow a 2nd-order predictor-corrector formulation implemented, see Fig. 10 (explicit scheme shown for simplicity). Since the structural mechanics solver is computationally less expensive than the fluid flow solver, such information can be taken advantage of in future extension of the code.

8. Results

A number of fluid flow simulations have been performed to validate the method presented in this work. Verifications were performed to ensure that the present method is second-order accurate on a single grid and in the presence of mesh refinement. The third and fourth-order Runge-Kutta time integrators are utilized for the shown results. Comparison has been made with numerical results reported in the literature. Unless otherwise noted, all simulations are conducted with $\gamma = 1.4$ and $Pr = 0.71$, and

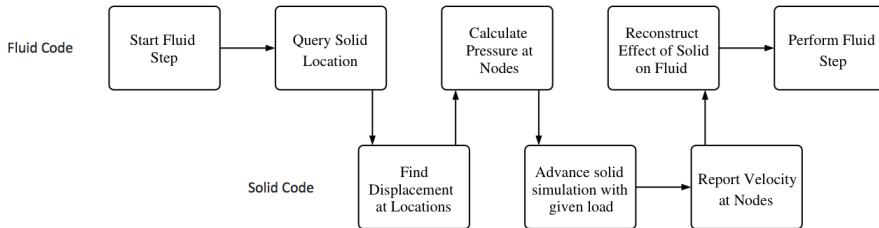


FIGURE 10. Example algorithm for fluid-structure interaction (explicit scheme).

$CFL \equiv u_\star \Delta t / \Delta x = 1/2$ unless otherwise noted. In what follows, the temporal variables reported in terms of convective time units.

8.1. Compressible Viscous Vortex

The low-Mach number flow induced by an initially-homentropic compressible viscous vortex (Colonius et al. 1991) is simulated in two dimensions at $M = 0.125$, $Re = 125$. A domain of size $[0, L]^2$ is used, with $L = 40r^\star$ and a spatial resolution of $N^2 = 800^2$ cells, and a fixed time step $\Delta t = 10^{-3}$. The vortex is centered at $(x^\star, y^\star) = (L/2, L/2)$ with an initial azimuthal velocity profile of

$$v_\theta(r) = \frac{ru_\star s_i}{r^\star} \exp\left(-\frac{(x-x^\star)^2 - (y-y^\star)^2}{4\mu t}\right), \quad (8.1)$$

where $u_\star$ denotes the maximum angular velocity, $r^\star$ is the radius at which the maximum angular velocity occurs, t is dimensional time, ρ_∞ , p_∞ , a_∞ , and μ_∞ are the far-field density, pressure, and sonic speed, and dynamic viscosity, respectively. Following Colonius et al. (1991), non-dimensional quantities may be defined as

$$\begin{aligned} \tilde{a} &= \frac{a}{a_\infty}, \quad M = \frac{v_m}{a_\infty}, \quad \tilde{v} = \frac{v}{v_m}, \quad \tilde{t} = \frac{v_m t}{r^\star} \\ Re &= \frac{\rho_\infty v_m r^\star}{\mu_\infty}, \quad \tilde{\tau} = \frac{\tilde{t}}{Re}, \quad \tilde{p} = \frac{p - p_\infty}{\rho_\infty v_m^2}, \quad \tilde{r} = \frac{r}{r^\star}. \end{aligned} \quad (8.2)$$

Comparison is made with the results of the high-order method presented in Colonius et al. (1991) for the purpose of validation. The pressure and tangential velocity profiles are shown in Fig. 11. Close agreement can be observed over the course of time as the vortex diffuses. It can be seen that the present solution agrees well spatially as well.

8.2. Two-Dimensional Turbulence in Bi-Periodic Domain

A two-dimensional, bi-periodic, unit domain is initialized to contain 50 Taylor vortices (Colonius et al. 1991) with randomly generated positions and strengths, from which an initial velocity field is calculated. Periodicity of the domain in the initial conditions is enforced by adding periodic images of the vortices in the domain.

Here the strength s_i is chosen from $[-1, 1]$ and the center (x_i, y_i) for vortex i are randomly chosen. Parameters are taken to be $u_\star = 0.1$, $\mu t = 0.00125$, and $r_\star = \sqrt{2\mu t}$. The Reynolds number based on $u_\star$ and $r_\star$ are set to 500 for the test below.

The equations for continuity and total energy are discretized in conservation form, (2.16) and (2.18), and refluxing is applied at mesh interfaces to ensure flux continuity. As such, mass and total energy are conserved for a bi-periodic domain, regardless of the level of mesh refinement.

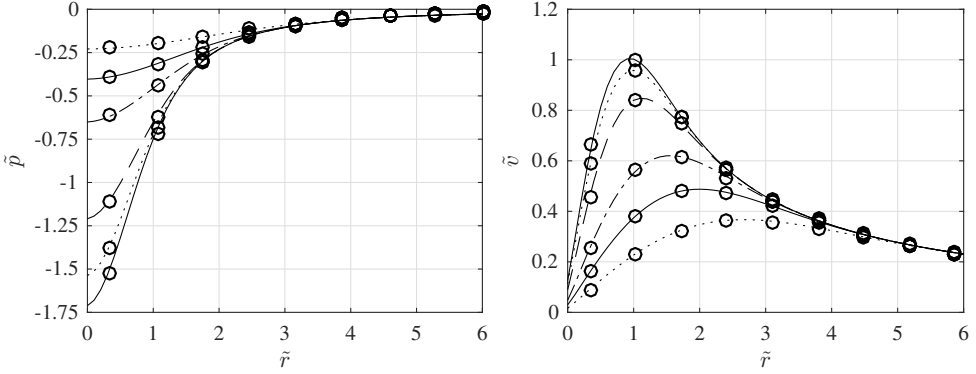


FIGURE 11. Non-dimensional pressure (left) and angular velocity (right) for the compressible, viscous vortex (Sec. 8.1). Results shown at non-dimensional times $\tilde{\tau} = 0.197$ (solid), $\tilde{\tau} = 0.217$ (dot), $\tilde{\tau} = 0.277$ (dash), $\tilde{\tau} = 0.517$ (dash-dot), $\tilde{\tau} = 0.837$ (solid), and $\tilde{\tau} = 1.477$ (dot). Open circles denote the solution from Colonius *et al.* (1991).

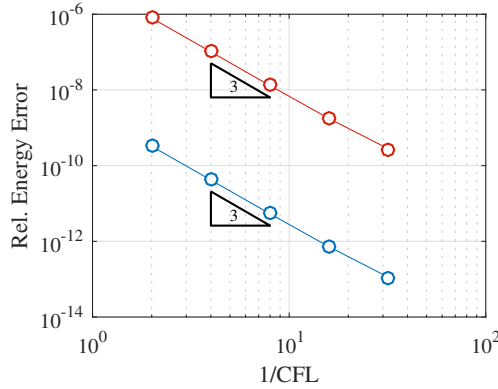


FIGURE 12. Relative errors for kinetic (red) and internal energy (blue), according to (8.5) for the bi-periodic vortex field on a single level of mesh refinement under temporal refinement. The magnitude of spurious energy transfer converges with the order of the selected time integrator (third-order Runge-Kutta method).

Figure 12 demonstrates that the temporal convergence of internal and kinetic energy is of the order of the time integrator, as discussed in Section 2.3. This result is novel, in that the work of Morinishi (2010) conserved total energy with the order of accuracy of the time integrator by simulating internal energy. This work simulates and exactly conserves total energy in the system, while limiting spurious transfer between derived quantities internal and kinetic energy by specifying the accuracy of the time integrator. Results are compiled at $\hat{t} = tu_\star/r_\star = 0.4$.

8.2.1. Adaptive Mesh Refinement

Figure 13 compares results of simulating a bi-periodic vorticity field using a single grid of uniform resolution of $N^2 = 512^2$ and a grid with two levels of dynamic AMR. The case with AMR is taken such that the finest mesh has the same effective mesh resolution as the single grid case. The refinement criterion is dependent on the vorticity magnitude. The case depicted in Fig. 13 is performed again using AMR, with a base resolution of $N^2 = 64^2$ and two levels of refinement, to demonstrate a variety of statistics and convergence results. Figure 14 shows relative mass and total energy errors in the presence of two levels of AMR, demonstrating conservation of mass and total energy up to machine

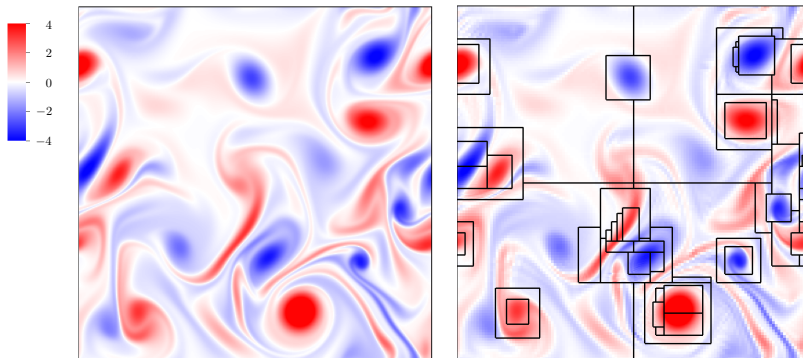


FIGURE 13. Comparison of the single-grid reference vorticity field (left) with the solution using two levels of AMR (right).

precision. Relative errors computed as

$$\frac{|\int_{\Omega^1} \rho^0 d\Omega - \int_{\Omega^1} \rho^N d\Omega|}{\int_{\Omega^1} \rho^0 d\Omega} \quad (8.3)$$

and

$$\frac{|\int_{\Omega^1} (\rho E)^0 d\Omega - \int_{\Omega^1} (\rho E)^N d\Omega|}{\int_{\Omega^1} (\rho E)^0 d\Omega}. \quad (8.4)$$

Superscripts represent time step. The evolution of the total number of cells on each level of mesh refinement is shown in Fig. 15, with a reference line showing the number of cells if a single grid of the same effective resolution were used. While there are some computational savings on mesh Ω^2 , which increase with time as vortex strength decays, the greatest savings are on the finest level, Ω^3 . On average, approximately 1/4 of the domain is resolved with the finest grid resolution for this example. We achieve computational savings over using a single fine grid over the entire domain by using dynamic AMR to track and resolve certain features of interest, while resolving the “bulk” fluid region using a coarse mesh. The runtimes for this problem with varying effective resolutions and levels of mesh refinement are summarized in Table 1. The savings from using AMR are more pronounced with a finer base mesh, as the computational overhead incurred by AMR routines is mitigated by the computational savings. Here, two grids are said to have the same effective resolution if they have the same finest level of resolution.

The fluctuation of relative internal energy error at various levels of mesh refinement (compared to results on a single grid of uniform mesh resolution), is depicted in Fig. 16. Energy exhibits small scale fluctuations, but does not induce “blow-up” in the solution. Relative error is computed as

$$\frac{|\int_{\Omega^1} (\rho e)^S d\Omega - \int_{\Omega^1} (\rho e)^{AMR} d\Omega|}{\int_{\Omega^1} (\rho e)^S d\Omega}. \quad (8.5)$$

Let superscript S denote results on a single grid and AMR denote results with mesh refinement upon termination of the simulation. One can observe that internal energy error in Fig. 16 increases with the number of levels of mesh refinement. This is due to the fact that the skew-symmetric property is not satisfied exactly at coarse/fine mesh boundaries. However, relative errors are still on the order of 10^{-5} . It should be noted that if a momentum-conservative central-differencing scheme is used for this test problem, the solution blows-up quickly.

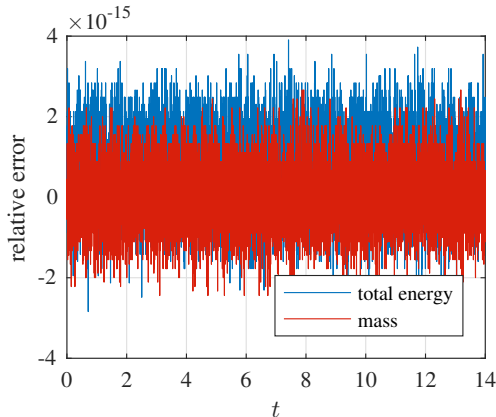


FIGURE 14. Relative errors for mass (red) and total energy (blue) (Eqs. (8.3) and (8.4), respectively) for the simulation of two-dimensional turbulence in a bi-periodic domain with 2 levels of dynamic AMR. Both quantities are conserved to machine precision.

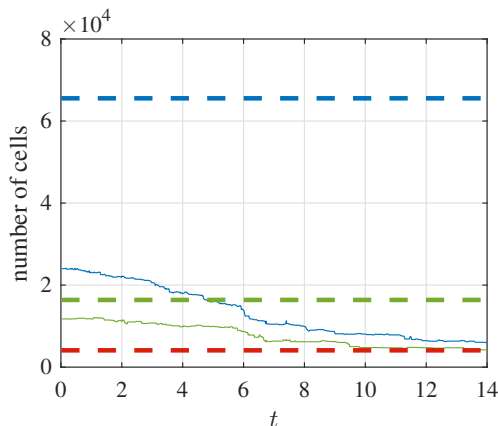


FIGURE 15. Number of grid cells versus time for the simulation shown in Fig. 13 with 2 levels of AMR. The solid line indicates the number of grid cells at a given level of resolution. The dashed line indicates the the number of cells required to mesh the domain with a single level of refinement. Level 1: red. Level 2: green. Level 3: blue.

8.3. 3D Taylor–Green Vortex/Turbulence

The velocity field on a three-dimensional tri-periodic domain $[0, 2\pi L]^3$ is initialized as a Taylor–Green vortex:

$$\begin{aligned} u &= U_0 \sin(x) \cos(y) \cos(z), & v &= -U_0 \cos(x) \sin(y) \cos(z), & w &= 0, \\ \rho &= 1, & p &= 100 + \frac{1}{16} [\cos(2x) + \cos(2y)] [\cos(2z) + 2]. \end{aligned}$$

Here, we take $U_0 = 1$ and $L = 1$, which is consistent with flow conditions at $M = 0.1$ for comparison with the literature. The viscosity is adjusted to perform the test at $Re_L = 1600$, where $Re_L = \rho U_0 L / \mu$. The flow field is visualized with the Q -criterion (Hunt *et al.* 1988) and is shown in Figure 17. The kinetic energy dissipation rate, calculated as

$$\frac{dK}{dt} = \frac{1}{\rho_0 |\Omega|} \int_{\Omega} \frac{1}{2} \rho \mathbf{u} \cdot \mathbf{u} d\Omega, \quad (8.6)$$

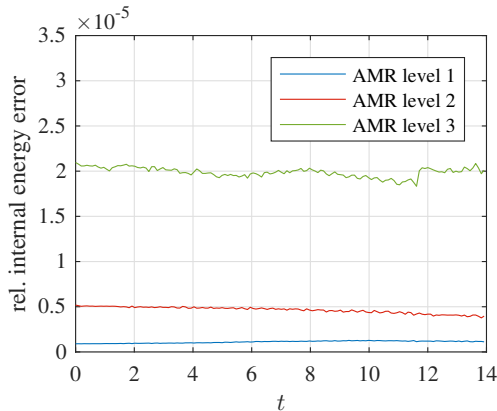


FIGURE 16. Relative internal energy error (compared to single grid) as in (8.5), for the the simulation of two-dimensional turbulence in a bi-periodic domain with AMR. Relative errors are shown at an effective mesh resolution of $N^2 = 256^2$, with 1 (blue, solid), 2 (red, dot), and 3 (green, dash-dot) levels of AMR (base resolutions of $N^2 = 32^2$ - 128^2).

Runtime (sec)			
Effective Resolution	Single Grid	Base +1 AMR Levels	Base +2 AMR Levels
128^2	27	51	59
256^2	234	179	172
512^2	3071	1102	731

TABLE 1. Single processor runtimes for the Sec. 8.2 test at varying grid resolution and levels of mesh refinement. We see that use of AMR incurs computational overhead that is mitigated for larger meshes. All tests are conducted with the same refinement criteria.

is reported in Fig. 18. The present results are compared with the results from Bull & Jameson (2015); DeBonis (2013); Gassner & Beck (2013); Van Rees *et al.* (2011) and show good agreement with increasing grid resolution.

We further demonstrate the energy convergence properties of the secondary-conservative, skew-symmetric, collocated method. The velocity field on a three-dimensional tri-periodic domain $[0, 2\pi L]^3$ is initialized as the summation of a set

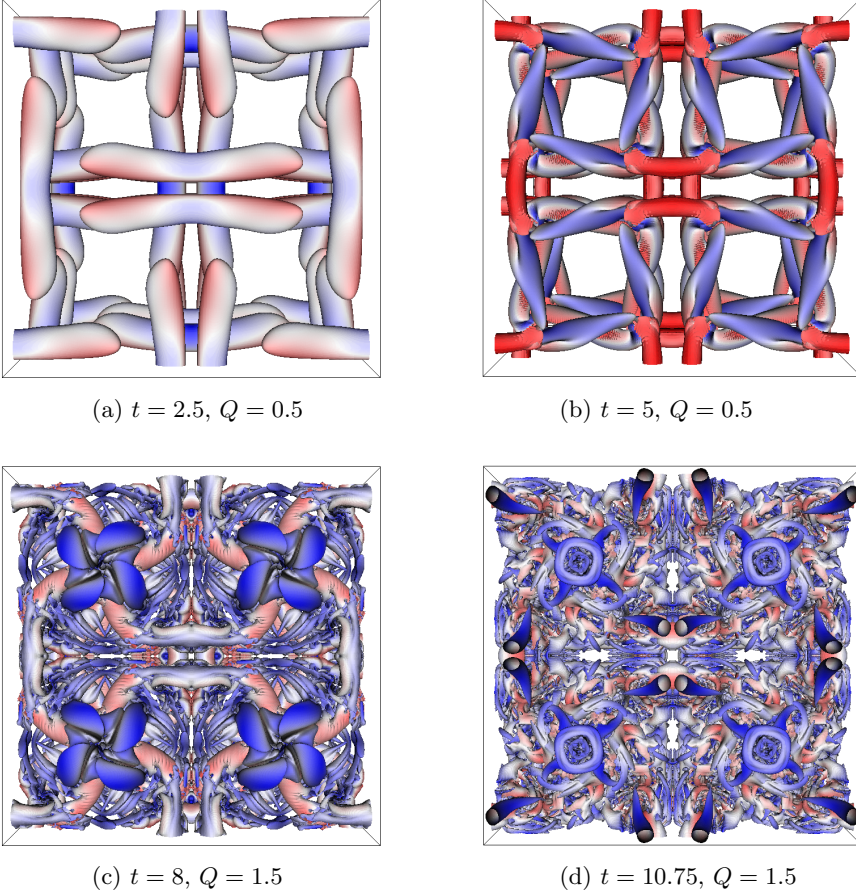


FIGURE 17. Evolution of Q -criterion for the Taylor–Green vortex simulation (Sec. 8.3), colored by velocity magnitude, show from $t = 2$ to 10.75. Results shown for of resolution $N^3 = 384^3$.

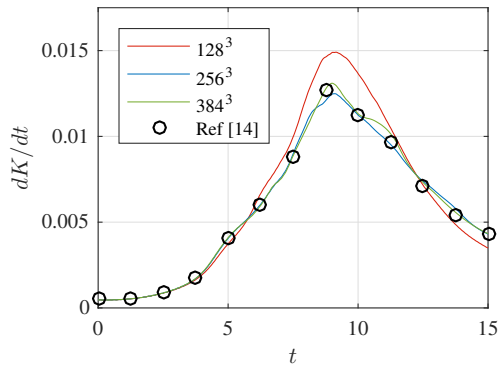


FIGURE 18. Kinetic energy dissipation rate according, Eq. (8.6), for the 3D Taylor–Green vortex simulation at resolutions $N^3 = 128^3$ (r), 256^3 (b), and 384^3 (g). Results compare favorably with reference solution (o), a 13-point, dispersion-relation preserving method (Bogey & Bailly 2004).

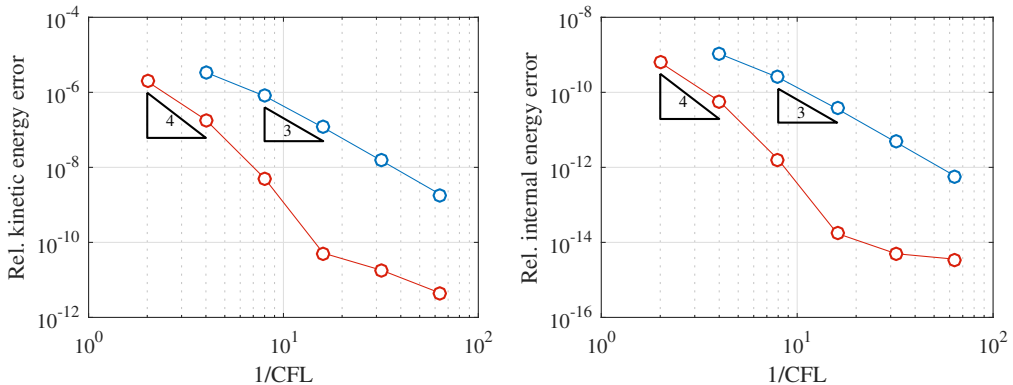


FIGURE 19. Relative errors for kinetic (left) and internal energy (right), based on (8.5), for the tri-periodic Taylor-Green vortex on a single level of mesh refinement under temporal refinement. The magnitude of spurious energy transfer converges with the order of the time integrator used, RK3 (blue) and RK4 (red). Degradation in convergence with RK4 at high resolution is due to the fact that the absolute error is on the order of machine precision.

of Taylor-Green vortices:

$$\begin{aligned}
 u &= U_0 \sum_{n=0}^2 c_n \sin(2^n x - \xi_n) \cos(2^n y - \gamma_n) \cos(2^n z - \zeta_n) + \delta_1(x, y, z), \\
 v &= -U_0 \sum_{n=0}^2 c_n \cos(2^n x - \xi_n) \sin(2^n y - \gamma_n) \cos(2^n z - \zeta_n) + \delta_2(x, y, z), \\
 w &= 0, \quad \rho = 1, \quad p = 100 + \frac{1}{16} [\cos(2x) + \cos(2y)] [\cos(2z) + 2].
 \end{aligned}$$

Here, δ_i represents a random perturbation to the velocity field to initiate turbulence, and $(\xi_n, \gamma_n, \zeta_n)$ are spatial offsets. The convective time unit is defined using $U_0 = 3$ and $L = 1$. The test is performed at Mach number $M = 0.3$, and Reynolds number $Re_L = 600$ to $t = 5$. Spatial resolution is $N^3 = 32^3$. The test is conducted using both a third- and fourth-order Runge-Kutta temporal integrators, to demonstrate that the temporal convergence of the kinetic and internal energy is equal to the order of the time integrator. To measure energy convergence, we run tests at a fixed time step with $CFL = 1/2^n$, for $n = 0, 1, 2, \dots, 6$ and measure the total amount of kinetic and internal energy in the domain. The proper convergence rates are observed as shown in Fig. 20. Total energy is conserved throughout all simulations to machine precision, as presented in Fig. 14.

8.4. Direct Numerical Simulation of 3D Isotropic Turbulence

Validation of the proposed method is demonstrated through the direct numerical simulation (DNS) of isotropic turbulence. The velocity field on a three-dimensional tri-periodic, unit domain is initialized as the sum of a number of periodic, solenoidal velocity fields with different frequencies, each given a random rotation and translation. The energy in each wave mode is distributed as in Morinishi (2010) and a periodic forcing term based on the velocity ($a\mathbf{u}$, for some constant a) is added to the right-hand side of the momentum equation. The test is run at Mach number $M = 0.1$ (with unit sound speed), and Reynolds number $Re_\lambda \approx 30$. Spatial resolution is fixed at $N = 512^3$. A third-order Runge-Kutta temporal integrator is used to advance the solution. Times are reported using $\tau = 2/(k_p U_0)$ (as in Bogey & Bailly (2004)) and $t_\eta = \sqrt{\nu L/U_0^3}$ (i.e., small eddy

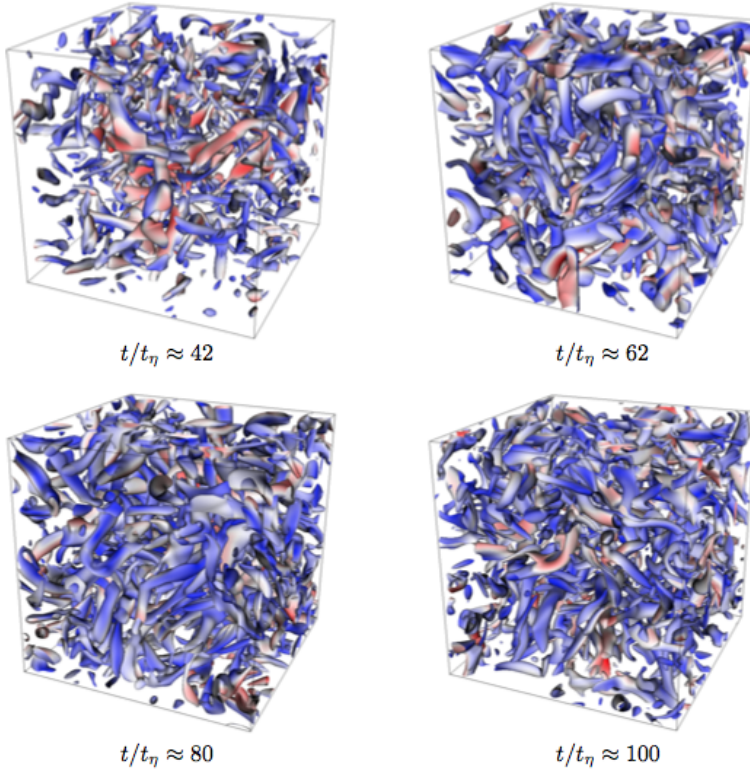


FIGURE 20. Simulation of forced isotropic turbulence. Isosurface of Q -criterion colored by velocity magnitude, at various eddy-turnover times.

timescale). This simulation was performed in parallel using MPI, run on 512 processors. Representative snapshots of the turbulent flow fields are shown in figure 20 which also exhibit the $-5/3$ kinetic energy decay characteristics.

8.5. Structural Mechanics

The thin plate model utilizes a geometry in which the thickness is less than one-tenth the width and length and assumes the resulting transverse displacements are small. To solve this problem numerically, we use a uniform grid, apply tensor cubic spline basis functions, and numerically integrate the PDE using four-point Gaussian quadrature.

The code was transferred from Matlab to FORTRAN code for integration into Boxlib. This code is comparable to prior routines in MATLAB suitable for nonlinear control designs (Oates & Smith 2008). Challenges in transitioning this code to FORTRAN subroutines capable of coupling to the fluid solver were addressed for seamless coupling with the fluid. This included development of appropriate methods and libraries for efficiently communicating data between the structure and fluid solvers.

Several verification cases were performed on the thin-plate model. As shown in Figure 21, we considered uniform loading across the entire surface of the plate and found agreement with plate theory. Importantly, we considered the case of varying load along the length of the plate where excellent agreement with plate theory is illustrated in Figure 21(d).

Lastly, a nonlinear shell was implemented to identify the necessary steps to couple large

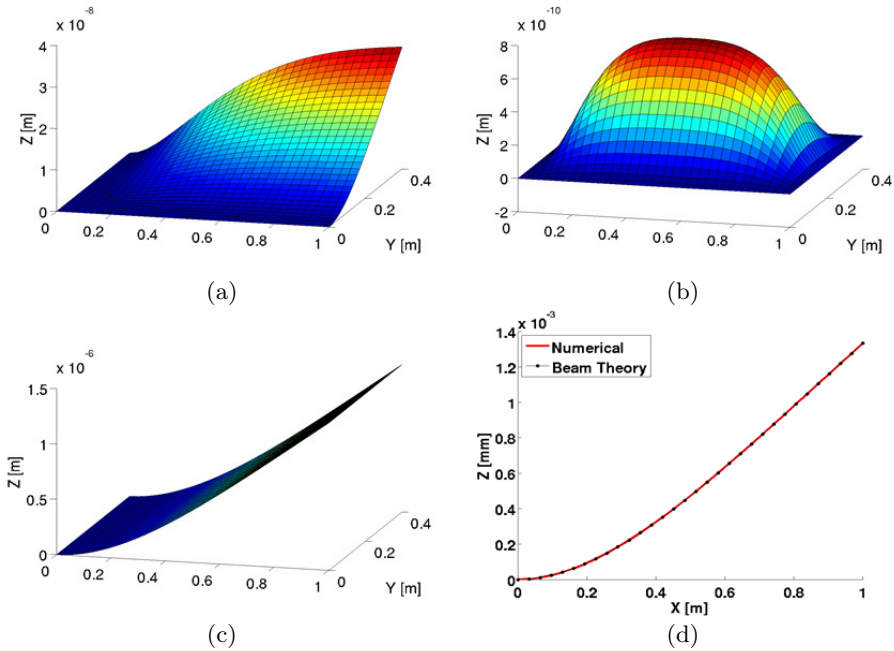


FIGURE 21. A thin plate model was implemented in order to test the interaction between the fluid and solid simulations on a small scale and computationally efficient problem. Several different boundary conditions were tested (a-c) and validation was performed on several load cases well documented in beam theory (d uniform loading across surface).

deformation nodal displacements and velocity outputs of the shell surface deformation for coupling with the fluid solver. This modeling approach allows for more complex geometry as well as large deformation using a fully conservative formulation. An example of this model is shown in Figure 22 where a hollow cylinder is deformed from transverse loading. True displacements are shown in (c) and (d) to highlight significant nonlinear geometric effects from loading. The procedure to transfer nodal degrees of freedom from the structure to the fluid follow directly as in the small displacement plate model.

8.6. Fluid-Structure Interaction

Solver for fluid-structure interaction is incorporated in present code with full nonlinear structural mechanics routines are currently being implemented. While extensive testing is still ongoing, we show here simple demonstration of vortex-induced vibration (VIV) captured for $Re = 110$. In this example, the cylinder is attached to a linear spring-mass-damper systems (Anagnostopoulos & Bearman 1992). The shown results in Fig. 23 are in agreement with experimental results. Similarly, we consider flapping of a rigid plate pinned at the leading edge at low Reynolds number of $Re = 200$. The snapshots of the flow fields from Fig. 24 demonstrate the the current solver is capable to dynamically allocating mesh refinement to capture the fluid-structure interaction. Future work will tackle the fluid-structure interaction capabilities for three-dimensional bodies with nonlinear-structure mechanics being interfaced with the fluid physics.

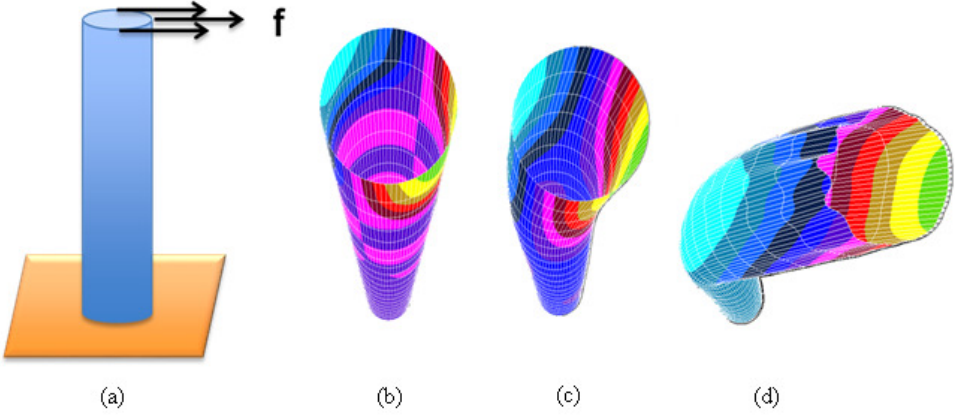


FIGURE 22. Additional research was conducted to implement a nonlinear shell model in order to allow for more complex geometry and large deformation while using a fully conservative formulation. This was implemented on a hollow cylindrical beam with large transverse loading as shown in (a). The undeformed beam is shown in (b) and different stages of deformation are shown in (c) and (d).

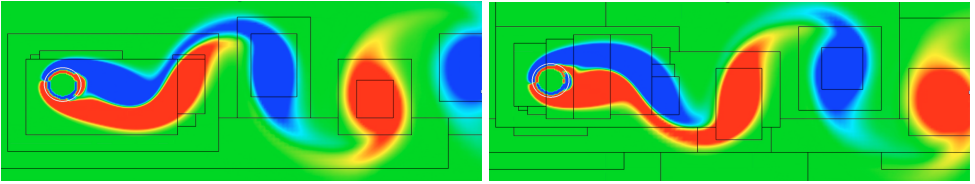


FIGURE 23. Testing of fluid-structure interaction capability for vortex-induced vibration of cylinder at $Re = 110$ with dynamic AMR.

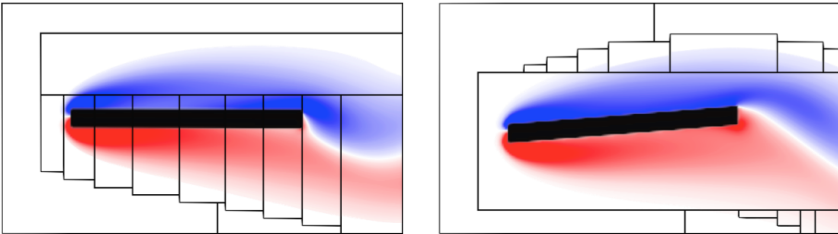


FIGURE 24. Testing of fluid-structure interaction capability for vortex-induced flapping of flat plate at $Re = 200$ with dynamic AMR.

9. Resulting Publications

- Canuto, D. and Taira, K., “Two-dimensional compressible viscous flow around a circular cylinder,” *Journal of Fluid Mechanics*, 785, 349-371, 2015.
- Lacis, U., Bagheri, S., and Taira, K., “A stable fluid-structure-interaction solver for low-density rigid bodies using the immersed boundary projection method,” *Journal of Computational Physics*, 305, 300-318, 2016.
- Jemison, M. and Taira, K., “A collocated, energy-preserving method for low-Mach number compressible flow,” in preparation, 2016.
- Miles, P., Hays, M., Smith, R., Oates, W., “Bayesian Uncertainty Analysis of Finite Deformation Viscoelasticity,” *Mechanics of Materials*, v. 91(1), p. 35-49, 2015.

- Miles, P., Guettler, A., Hussaini, M.Y., Oates, W. “Uncertainty analysis of dielectric elastomer membranes under multiaxial loading,” ASME Conference on Smart Materials, Adaptive Structures and Intelligent Systems (SMASIS), (student best paper award), 2015.
- Miles, P., Hays, M., Smith, R., Oates, W. “Uncertainty Analysis of a Finite Deformation Viscoelastic Model,” ASME Conference on Smart Materials, Adaptive Structures and Intelligent Systems (SMASIS), 2015.
- Miles, P., Gao, W., Moura, A., Hussaini, M.Y., Oates, W., “Uncertainty Analysis of Dielectric Elastomer Membranes Under Electromechanical Loading,” in preparation, 2016.

10. Participants

- Kunihiko Taira, Assistant Professor (FAMU/FSU)
- William S. Oates, Associate Professor (FAMU/FSU)
- Matthew Jemison, Postdoctoral Research Associate (FAMU/FSU)
→ Currently with Naval Surface Warfare Center (Carderock)
- Paul Miles, Graduate Research Assistant (FAMU/FSU)
- Ryan Jantzen, Graduate Research Assistant (FAMU/FSU)
→ Currently with Air Force Research Laboratory (Eglin AFB)
- Daniel Canuto, Undergraduate Research Assistant (FAMU/FSU)
→ Currently with University of California, Los Angeles
- Peter Van Brussel, Undergraduate Research Assistant (FAMU/FSU)

11. Remarks and Future Work

This project has produced a fluid-structure interaction simulation code for compressible flow. The code is written to leverage consistent finite different schemes for the full Navier-Stokes equations with desirable conservation properties. The code is written with adaptive mesh refinement capabilities using the BoxLib library and is able to strategically allocate the computational resource towards fine scale vortices in the flow field. The canonical 3D Taylor vortex example and isotropic turbulence simulations have highlighted these features.

Moreover, we have implemented an immersed boundary method based on the level set formulation. This enables the simulation of flows over bodies of any shape without the requirement of re-meshing the flow field at each time step (for moving or deforming bodies). The current program has the unique capabilities to handle sharp corners, which is often present in aeronautics. An example of simulating flows around such body is shown in Fig. 25. The present code is able to predict the emergence of the flow separation.

The structural mechanics solver for linear and nonlinear dynamics has also been developed and is now merged with the fluid flow solvers. At the moment, the verification and validation for fluid-structure interaction has been performed for rigid-body interactions and those with deformations are currently underway. The combination of such analysis for three-dimensional flow will be performed in the future and will also be examined for high-Reynolds number flow problems.

The supported effort here has led to the development of a significant programming code that can be used for a wide range of problems. The user guide is attached at the end of this report as an appendix. We also note that two researchers supported under this grant have joined the Department of Defense research laboratories (Naval Surface Warfare Center and Air Force Research Laboratory).

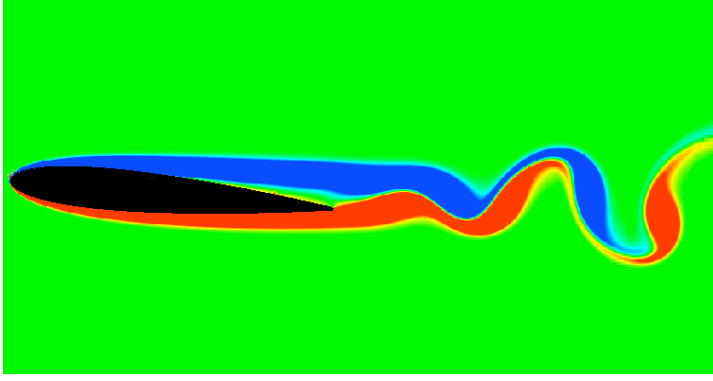


FIGURE 25. Separated flow (vorticity field) over a NACA0012 airfoil at $Re = 5000$ and 5 degrees angle of attack. The shown result uses 4 level of mesh refinement.

12. Acknowledgements

We would like to thank Dr. Ann Almgren, Dr. Andy Nonaka, and Dr. Mark Sussman for their guidance on the use of BoxLib.

Appendix A. Discrete Tensor Operator Conservation

The collocated tensor discretization (2.24) is specially chosen to remain simultaneously consistent with the total, internal, and kinetic energy equations, while conserving energy in the sense of Morinishi. This property will be demonstrated here in 2D. The extension to 3D is straightforward.

$$\tau_{ij} = \begin{pmatrix} \frac{4}{3}\mu\delta_x(\bar{u}^x) - \frac{2}{3}\mu\delta_y(\bar{v}^y) & \mu(\delta_y(\bar{u}^y) + \delta_x(\bar{v}^x)) \\ \mu(\delta_y(\bar{u}^y) + \delta_x(\bar{v}^x)) & \frac{4}{3}\mu\delta_y(\bar{v}^y) - \frac{2}{3}\mu\delta_x(\bar{u}^x) \end{pmatrix} \quad (\text{A } 1)$$

We will use (A 1) to write the kinetic dissipation terms in the discrete kinetic energy equation (2.19), obtained by multiplying the momentum equation by velocity and summing over all components.

$$\begin{aligned} u_i\delta_{x_j}(\overline{\tau_{ij}^{xj}}) &= u \left(\delta_x \left(\frac{4}{3}\mu\delta_x(\bar{u}^x) - \frac{2}{3}\mu\delta_y(\bar{v}^y) \right) + \delta_y \left(\mu(\delta_y(\bar{u}^y) + \delta_x(\bar{v}^x)) \right) \right) + \\ &v \left(\delta_x \left(\mu(\delta_y(\bar{u}^y) + \delta_x(\bar{v}^x)) \right) + \delta_y \left(\frac{4}{3}\mu\delta_y(\bar{v}^y) - \frac{2}{3}\mu\delta_x(\bar{u}^x) \right) \right) = \\ &u(\delta_x(\overline{\tau_{11}^x}) + \delta_y(\overline{\tau_{12}^y})) + v(\delta_x(\overline{\tau_{21}^x}) + \delta_y(\overline{\tau_{22}^y})) \end{aligned} \quad (\text{A } 2)$$

Now, we will rewrite the viscous heating term from the total energy equation (2.18).

$$\delta_{x_j}(\widetilde{u_i\tau_{ij}^{xj}}) = \delta_x(\widetilde{u\tau_{11}^x}) + \delta_y(\widetilde{u\tau_{12}^y}) + \delta_x(\widetilde{v\tau_{21}^x}) + \delta_y(\widetilde{v\tau_{22}^y}) \quad (\text{A } 3)$$

Then, we can use the discrete forms (A 2-A 3), along with identity (2.13) to recover the viscous stress term appearing in the discrete internal energy equation (2.20).

$$\delta_{x_j}(\widetilde{\tau_{ij}u_i^{xj}}) - u_i\delta_{x_j}(\overline{\tau_{ij}^{xj}}) = \tau_{ij}\delta_{x_j}(\widetilde{u_i^{xj}}) \quad (\text{A } 4)$$

Consider now the standard 9-point stencil for the term $\delta_{x_j}(\tau_{ij})$ appearing in the momentum equation. The corresponding face-centered tensor discretization can be written as

$$\tau_{ij} = \begin{pmatrix} \bar{\mu}^x \left(\frac{4}{3} \delta_x(u) - \frac{2}{3} \overline{\delta_y(\bar{v}^y)}^x \right) & \bar{\mu}^x \left(\overline{\delta_y(\bar{u}^y)}^x + \delta_x(v) \right) \\ \bar{\mu}^y \left(\overline{\delta_x(\bar{v}^x)}^y + \delta_y(u) \right) & \bar{\mu}^y \left(\frac{4}{3} \delta_y(v) - \frac{2}{3} \overline{\delta_x(\bar{u}^x)}^y \right) \end{pmatrix}. \quad (\text{A } 5)$$

It is clear that if the tensor terms were discretized at faces, to utilize the 9-point 2D stencil in the momentum equation (2.17) and (A 2), we would not be able to use the discrete commutativity property to obtain a discrete equation that is consistent with the internal energy equation.

REFERENCES

- ALMGREN, AS, BECKNER, VE, BELL, JB, DAY, MS, HOWELL, LH, JOGGERST, CC, LIJEWSKI, MJ, NONAKA, A, SINGER, M & ZINGALE, M 2010a CASTRO: A new compressible astrophysical solver. I. hydrodynamics and self-gravity. *The Astrophysical Journal* **715** (2), 1221.
- ALMGREN, A. S., BECKNER, V.E., BELL, J. B., DAY, M.S., HOWELL, L.H., JOGGERST, C. C., LIJEWSKI, M.J., NONAKA, A., SINGER, M. & ZINGALE, M. 2010b Castro: A new compressible astrophysical solver. i. hydrodynamics and self-gravity. *Astrophysical Journal* **715**, 1221–1238.
- ALMGREN, ANN S., BELL, JOHN B., COLELLA, PHILLIP, HOWELL, LOUIS H. & WELCOME, MICHAEL L. 1998 A conservative adaptive projection method for the variable density incompressible navier-stokes equations. *J. Comput. Phys* **142**, 1–46.
- ALMGREN, ANN S, BELL, JOHN B, LIJEWSKI, MIKE J, LUKIĆ, ZARIJA & VAN ANDEL, ETHAN 2013 Nyx: A massively parallel AMR code for computational cosmology. *The Astrophysical Journal* **765** (1), 39.
- ALMGREN, A S, BELL, J B & ZINGALE, M 2007 MAESTRO: A low Mach number stellar hydrodynamics code. *Journal of Physics: Conference Series* **78** (1), 012085.
- ANAGNOSTOPOULOS, P. & BEARMAN, P. W. 1992 Response characteristics of a vortex-excited cylinder at low Reynolds numbers. *J. Fluids. Struc.* **6**, 39–50.
- ANDERSON, W KYLE & BONHAUS, DARYL L 1994 An implicit upwind algorithm for computing turbulent flows on unstructured grids. *Computers & Fluids* **23** (1), 1–21.
- BADEN, SCOTT B 2000 *Structured adaptive mesh refinement (SAMR) grid methods*, , vol. 117. Springer Science & Business Media.
- BELL, J, ALMGREN, A, BECKNER, V, DAY, M, LIJEWSKI, M, NONAKA, A & ZHANG, W 2012 Boxlib user's guide. *Tech. Rep.*. Technical Report, CCSE, Lawrence Berkeley National Laboratory. Available at: <https://ccse.lbl.gov/BoxLib/BoxLibUsersGuide.pdf>.
- BERGER, M & LEVEQUE, R 1998a Adaptive mesh refinement for two-dimensional hyperbolic systems and the AMRCLAW software. *SIAM J. Numer. Anal* **35**, 2298–2316.
- BERGER, MARSHA J. 1987 On conservation at grid interfaces. *SIAM Journal on Numerical Analysis* **24** (5), 967–984, iCASE Report No. 84-43, September, 1984.
- BERGER, M. J. & COLELLA, P. 1989 Local adaptive mesh refinement for shock hydrodynamics. *Journal of Computational Physics* **82**, 64–84.
- BERGER, MARSHA J. & LEVEQUE, RANDALL J. 1998b Adaptive mesh refinement using wave-propagation algorithms for hyperbolic systems. *SIAM Journal on Numerical Analysis* **35** (6), 2298–2316.
- BLACKBURN, HUGH & HENDERSON, RON 1996 Lock-in behavior in simulated vortex-induced vibration. *Experimental Thermal and Fluid Science* **12** (2), 184–189.
- BOGEY, CHRISTOPHE & BAILLY, CHRISTOPHE 2004 A family of low dispersive and low dissipative explicit schemes for flow and noise computations. *Journal of Computational Physics* **194** (1), 194–214.
- BREUER, MICHAEL 1998 Large eddy simulation of the subcritical flow past a circular cylinder: numerical and modeling aspects. *International Journal for Numerical Methods in Fluids* **28** (9), 1281–1302.
- BROOKS, ALEXANDER N & HUGHES, THOMAS JR 1982 Streamline upwind/ Petrov-

- galerkin formulations for convection dominated flows with particular emphasis on the incompressible navier-stokes equations. *Computer methods in applied mechanics and engineering* **32** (1), 199–259.
- BROUWER, J., REISS, J. & SESTERHENNN, J. 2014 Conservative time integrators of arbitrary order for skew-symmetric finite-difference discretizations of compressible flow. *Computers and Fluids* **100**, 1–12.
- BULL, JR & JAMESON, A 2015 Simulation of the compressible Taylor–Green vortex using high-order flux reconstruction schemes. *AIAA Journal* **53**.
- CHEN, S., MERRIMAN, B., OSHER, S. & SMEREKA, P. 1997 A simple level set method for solving stefan problems. *J. Comp. Phys.* **135** (1), 8–29.
- COLELLA, P, GRAVES, DT, LIGOCKI, TJ, MARTIN, DF, MODIANO, D, SERAFINI, DB & VAN STRAALEN, B 2000 Chombo software package for AMR applications-design document.
- COLONIUS, TIM, LELE, SANJIVA K & MOIN, PARVIZ 1991 The free compressible viscous vortex. *Journal of Fluid Mechanics* **230**, 45–73.
- CRUTCHFIELD, WILLIAM Y. & WELCOME, MICHAEL L. 1993 Object-oriented implementation of adaptive mesh refinement algorithms. *Sci. Program.* **2** (4), 145–156.
- DEBONIS, JAMES 2013 Solutions of the Taylor–Green vortex problem using high-resolution explicit finite difference methods. AIAA Paper 2013-0382.
- DEITERDING, RALF 2003 AMROC-block structured adaptive mesh refinement in object-oriented C++.
- DESJARDINS, O., BLANQUART, G., BALARAC, G. & PITSCH, H. 2008 High order conservative finite difference scheme for variable density low mach number turbulent flows. *J. Comput. Phys.* **227** (15), 7125–7159.
- DOMMERMUTH, DOUGLAS G, O’SHEA, THOMAS T, WYATT, DONALD C, SUSSMAN, MARK, WEYMOUTH, GABRIEL D, YUE, DICK KP, ADAMS, PAUL & HAND, RANDALL 2014 The numerical simulation of ship waves using cartesian-grid and volume-of-fluid methods. *arXiv preprint arXiv:1410.1937* .
- ERICKSON, DAVID 2005 Towards numerical prototyping of labs-on-chip: modeling for integrated microfluidic devices. *Microfluidics and Nanofluidics* **1** (4), 301–318.
- ERICKSON, DAVID & LI, DONGQING 2004 Integrated microfluidic devices. *Analytica Chimica Acta* **507** (1), 11–26.
- FALTINSEN, ODD 1993 *Sea loads on ships and offshore structures*, , vol. 1. Cambridge university press.
- FERM, LARS & LÖTSTEDT, PER 2004 Accurate and stable grid interfaces for finite volume methods. *Appl. Numer. Math.* **49** (2), 207–224.
- FERZIGER, J. H. & PERIC, M. 1999 *Computational Methods for Fluid Dynamics*. Berlin: Springer.
- GASSNER, GREGOR J & BECK, ANDREA D 2013 On the accuracy of high-order discretizations for underresolved turbulence simulations. *Theoretical and Computational Fluid Dynamics* **27** (3-4), 221–237.
- GRIFFITH, BOYCE E., HORNING, RICHARD D., MCQUEEN, DAVID M. & PESKIN, CHARLES S. 2007 An adaptive, formally second order accurate version of the immersed boundary method. *J. Comput. Phys.* **223** (1), 10–49.
- GURTIN, M. 1967 Thermodynamics with internal state variables. *J. Chem. Phys.* **47**, 597–613.
- HAYS, M., MORTON, J., DICKINSON, B., CHAKRAVARTY, U. & OATES, W. 2013 Aerodynamic control of micro air vehicle wings using electroactive membranes. *J. Intell. Mater. Syst. Struct.* **24** (7), 862–878.
- HENDERSON, RONALD D & KARNIADAKIS, GEORGE EM 1995 Unstructured spectral element methods for simulation of turbulent flows. *Journal of Computational Physics* **122** (2), 191–217.
- HOLZAPFEL, GERHARD A. & SIMO, JUAN C. 1996 A new viscoelastic constitutive model for continuous media at finite thermomechanical changes. *Int. J. Solids Struct.* **33** (2022), 3019 – 3034.
- HONEIN, ALBERT E. & MOIN, PARVIZ 2004 Higher entropy conservation and numerical stability of compressible turbulence simulations. *J. Comput. Phys.* **201** (2), 531–545.
- HOU, THOMAS Y., LI, ZHILIN, OSHER, STANLEY & ZHAO, HONGKAI 1997 A hybrid

- method for moving interface problems with application to the hele-shaw flow. *J. Comp. Phys.* **134**, 236–252.
- HUNT, J. C. R., WRAY, A. A. & MOIN, P. 1988 Eddies, strems, and convergence zones in turbulent flows. *Tech. Rep.*. Center for Turbulence Research.
- ILINCA, F & HÉTU, J-F 2011 A finite element immersed boundary method for fluid flow around rigid objects. *International Journal for Numerical Methods in Fluids* **65** (7), 856–875.
- JAMESON, ANTHONY & MAVRIPLIS, D 1986 Finite volume solution of the two-dimensional euler equations on a regular triangular mesh. *AIAA journal* **24** (4), 611–618.
- JOHNSON, FORRESTER T, TINOCO, EDWARD N & YU, N JONG 2005 Thirty years of development and application of cfd at boeing commercial airplanes, seattle. *Computers & Fluids* **34** (10), 1115–1151.
- KALITZIN, GEORGI & IACCARINO, GIANLUCA 2003 Toward immersed boundary simulation of high reynolds number flows. *Tech. Rep.*. DTIC Document.
- KARNIADAKIS, GEORGE, BESKOK, ALI & ALURU, NARAYAN 2006 *Microflows and nanoflows: fundamentals and simulation*, , vol. 29. Springer Science & Business Media.
- KARNIADAKIS, GEORGE & SHERWIN, SPENCER 2013 *Spectral/hp element methods for computational fluid dynamics*. Oxford University Press.
- KHALIGHI, YASER, MANI, ALI, HAM, FRANK & MOIN, PARVIZ 2010 Prediction of sound generated by complex flows at low mach numbers. *AIAA journal* **48** (2), 306–316.
- KIM, DONGJOO & CHOI, HAECHAEON 2000 A second-order time-accurate finite volume method for unsteady incompressible flow on hybrid unstructured grids. *Journal of computational physics* **162** (2), 411–428.
- KOFOD, GUGGI, SOMMER-LARSEN, PETER, KORNBLUH, ROY & PELRINE, RON 2003 Actuation response of polyacrylate dielectric elastomers. *Journal of intelligent material systems and structures* **14** (12), 787–793.
- KRAMER, R. M. J., PANTANO, CARLOS & PULLIN, D. I. 2009 Nondissipative and energy-stable high-order finite-difference interface schemes for 2-d patch-refined grids. *J. Comput. Physics* **228** (14), 5280–5297.
- LIU, YEN, VINOKUR, MARCEL & WANG, ZJ 2006 Spectral difference method for unstructured grids i: basic formulation. *Journal of Computational Physics* **216** (2), 780–801.
- MACNEICE, PETER, OLSON, KEVIN M, MOBARRY, CLARK, DE FAINCSTEIN, ROSALINDA & PACKER, CHARLES 2000 Paramesh: A parallel adaptive mesh refinement community toolkit. *Computer physics communications* **126** (3), 330–354.
- MALVERN, L. 1969 *Introduction to the Mechanics of a Continuous Medium*. Englewood Cliffs, NJ: Prentice-Hall, Inc.
- MARTINEAU, RICHARD C & BERRY, RAY A 2004 The pressure-corrected ice finite element method for compressible flows on unstructured meshes. *Journal of Computational Physics* **198** (2), 659–685.
- MCCORQUODALE, PETER & COLELLA, PHILLIP 2011 A high-order finite-volume method for conservation laws on locally refined grids. *Communications in Applied Mathematics and Computational Science* **6** (1), 1–25.
- MILES, PAUL, GUETTLER, ADRIANE, HUSSAINI, YOUSUFF & OATES, WILLIAM S 2015a Uncertainty analysis of dielectric elastomer membranes under multi-axial loading. In *ASME 2015 Conference on Smart Materials, Adaptive Structures and Intelligent Systems*, pp. V001T02A004–V001T02A004. American Society of Mechanical Engineers.
- MILES, PAUL, HAYS, MICHAEL, SMITH, RALPH & OATES, WILLIAM 2015b Bayesian uncertainty analysis of finite deformation viscoelasticity. *Mechanics of Materials* **91**, 35–49.
- MILES, P., HAYS, M., SMITH, R. & OATES, W. 2015c Uncertainty analysis of a finite deformation viscoelasticity model. *Mechanics of Materials* **91**, 35–49.
- MITTAL, R., DONG, H., BOZKURTAS, M., NAJJAR, F. M., VARGAS, A. & VON LOEBBECKE, A. 2008 A versatile sharp interface immersed boundary method for incompressible flows with complex boundaries. *J. Comput. Phys.* **227** (10), 4825–4852.
- MITTAL, RAJAT & IACCARINO, GIANLUCA 2005 Immersed boundary methods. *Annu. Rev. Fluid Mech.* **37**, 239–261.
- MITTAL, SANJAY & TEZDUYAR, TAYFUN E 1994 Massively parallel finite element computation of incompressible flows involving fluid-body interactions. *Computer Methods in Applied Mechanics and Engineering* **112** (1), 253–282.

- MOHAMMAD, ABRAR H, WANG, ZJ & LIANG, CHUNLEI 2010 Large eddy simulation of flow over a cylinder using high-order spectral difference method. *Adv. Appl. Math. Mech* **2** (4), 451–466.
- MORINISHI, Y. 2010 Skew-symmetric form of convective terms and fully conservative finite difference schemes for variable density low-mach number flows. *J. Comput. Phys.* **229** (2), 276–300.
- MORINISHI, Y. & KOGA, K. 2014 Skew-symmetric convection form and secondary conservative finite difference methods for moving grids. *Journal of Computational Physics* **257**, 1081–1112.
- MORINISHI, Y., LUND, T. S., VASILYEV, O. V. & MOIN, P. 1998 Fully conservative higher order finite difference schemes for incompressible flow. *J. Comput. Phys.* **143** (1), 90–124.
- MUELLER, THOMAS J 2001 *Fixed and flapping wing aerodynamics for micro air vehicle applications*, , vol. 195. AIAA.
- NORRIS, SE, WERE, CJ, RICHARDS, PJ & MALLINSON, GD 2011 A voronoi-based ale solver for the calculation of incompressible flow on deforming unstructured meshes. *International Journal for Numerical Methods in Fluids* **65** (10), 1160–1179.
- OATES, W.S., EVANS, P., SMITH, R.C. & DAPINO, M. 2009 Experimental implementation of a hybrid nonlinear control design for magnetostrictive transducers. *J. Dyn. Syst.-T. ASME* **131**, 041004 (11 pages).
- OATES, W.S. & SMITH, R.C. 2008 Nonlinear optimal control techniques for vibration attenuation using nonlinear magnetostrictive actuators. *J. Intell. Mater. Sys. Struct.* **19** (2), 193–209.
- OSHER, STANLEY & SETHIAN, JAMES A. 1988 Fronts propagating with curvature dependent speed: Algorithms based on hamilton-jacobi formulations. *J. Comp. Phys.* **79** (1), 12–49.
- PANTANO, CARLOS, DEITERDING, R., HILL, D. J. & PULLIN, D. I. 2007 A low numerical dissipation patch-based adaptive mesh refinement method for large-eddy simulation of compressible flows. *J. Comput. Physics* **228** (221), 63–87.
- PENG, DANPING, MERRIMAN, BARRY, OSHER, STANLEY, ZHAO, HONGKAI & KANG, MYUNGJOO 1999 A pde-based fast local level set method. *Journal of Computational Physics* **155**.
- PENG, S., VALANIS, K. & LANDEL, R. 1977 Nonlinear viscoelasticity and relaxation phenomena of polymer solids. *Acta Mechanica* **25**, 229–240.
- PERI, DANIELE, ROSSETTI, MICHELE & CAMPANA, EMILIO F 2001 Design optimization of ship hulls via cfd techniques. *Journal of Ship Research* **45** (2), 140–149.
- PESKIN, C.S. 1977 Numerical analysis of blood flow in the heart. *Journal of Computational Physics* **25** (3), 220–252.
- PESKIN, CHARLES S. 2002 The immersed boundary method. *Acta Numerica* **11**, 479–517.
- PIERCE, C. D. 2001 Progress-variable approach for large-eddy simulation of turbulent combustion. PhD thesis, Stanford University.
- PINES, DARRYL J & BOHORQUEZ, FELIPE 2006 Challenges facing future micro-air-vehicle development. *Journal of aircraft* **43** (2), 290–305.
- POPE, STEPHEN B 2000 *Turbulent flows*. Cambridge university press.
- RAMAMURTI, RAVI, SANDBERG, WILLIAM & LÖHNER, RAINALD 2000 Simulation of the dynamics of micro air vehicles. In *AIAA Aerospace Sciences Meeting and Exhibit*, pp. 10–13.
- RHIE, CM & CHOW, WL 1983 Numerical study of the turbulent flow past an airfoil with trailing edge separation. *AIAA journal* **21** (11), 1525–1532.
- RIZZELLO, GIANLUCA, HODGINS, MICAH, NASO, DAVID, YORK, ALEXANDER & SEELECKE, STEFAN 2014 Dynamic electromechanical modeling of a spring-biased dielectric electroactive polymer actuator system. In *ASME 2014 Conference on Smart Materials, Adaptive Structures and Intelligent Systems*, pp. V002T02A012–V002T02A012. American Society of Mechanical Engineers.
- ROMA, ALEXANDRE M., PESKIN, CHARLES S. & BERGER, MARSHA J. 1999 An adaptive version of the immersed boundary method. *J. Comput. Phys.* **153** (2), 509–534.
- SCHLICHTING, HERRMANN, GERSTEN, KLAUS & GERSTEN, KLAUS 2000 *Boundary-layer theory*. Springer Science & Business Media.
- SETHIAN, JA & SMEREKA, PETER 2003 Level set methods for fluid interfaces. *Annual Review of Fluid Mechanics* **35** (1), 341–372.

- SMEREKA, PETER 2006 The numerical approximation of a delta function with application to level set methods. *J. Comput. Phys.* **211** (1), 77–90.
- SMITH, R.C. 2005 *Smart Material Systems: Model Development*. Philadelphia, PA: SIAM.
- STEVENS, BRIAN L & LEWIS, FRANK L 2003 *Aircraft control and simulation*. John Wiley & Sons.
- SUBBAREDDY, PRAMOD K. & CANDLER, GRAHAM V. 2009 A fully discrete, kinetic energy consistent finite-volume scheme for compressible flows. *J. Comput. Phys.* **228** (5), 1347–1364.
- TEZDUYAR, TAYFUN E 2001 Finite element methods for flow problems with moving boundaries and interfaces. *Archives of Computational Methods in Engineering* **8** (2), 83–130.
- TEZDUYAR, TAYFUN E, TAKIZAWA, KENJI, MOORMAN, CREIGHTON, WRIGHT, SAMUEL & CHRISTOPHER, JASON 2010 Space-time finite element computation of complex fluid-structure interactions. *International Journal for Numerical Methods in Fluids* **64** (10-12), 1201–1218.
- TRITTON, DAVID J 2012 *Physical fluid dynamics*. Springer Science & Business Media.
- UDAYKUMAR, H. S., MITTAL, R., RAMPUNGOON, P. & KHANNA, A. 2001 A sharp interface cartesian grid method for simulating flows with complex moving boundaries. *J. Comput. Phys.* **174**, 345–380.
- UDDIN, H., KRAMER, R. M. J. & PANTANO, C. 2014 A Cartesian-based embedded geometry technique with adaptive high-order finite differences for compressible flow around complex geometries. *J. Comput. Phys.* **262** (??), 379–407.
- VAN REES, WIM M, LEONARD, ANTHONY, PULLIN, DI & KOUMOUTSAKOS, PETROS 2011 A comparison of vortex and pseudo-spectral methods for the simulation of periodic vortical flows at high Reynolds numbers. *Journal of Computational Physics* **230** (8), 2794–2805.
- VANELLA, MARCOS & BALARAS, ELIAS 2009 A moving-least-squares reconstruction for embedded-boundary formulations. *Journal of Computational Physics* **228** (18), 6617–6628.
- VANELLA, MARCOS, RABENOLD, PATRICK & BALARAS, ELIAS 2008 Effect of grid discontinuities on large-eddy simulation statistics and flow fields. *J. Turb.* **9** (32), 1–23.
- VANELLA, MARCOS, RABENOLD, PATRICK & BALARAS, ELIAS 2010 A direct-forcing embedded-boundary method with adaptive mesh refinement for fluid-structure interaction problems. *J. Comput. Physics* **229** (18), 6427–6449.
- WALL, C., PIERCE, C. D. & MOIN, P. 2002 A Semi-implicit Method for Resolution of Acoustic Waves in Low Mach Number Flows. *Journal of Computational Physics* pp. 545–563.
- WHITE, FRANK M. 1991 *Viscous fluid flow*. New York: McGraw-Hill.
- WINTERS, ANDREW R & KOPRIVA, DAVID A 2014 High-order local time stepping on moving dg spectral element meshes. *Journal of Scientific Computing* **58** (1), 176–202.
- WISSINK, ANDREW M, HORNUNG, RICHARD D, KOHN, SCOTT R, SMITH, STEVE S & ELLIOTT, NOAH 2001 Large scale parallel structured AMR calculations using the SAMRAI framework. In *Supercomputing, ACM/IEEE 2001 Conference*, pp. 22–22. IEEE.
- WISSLER, MICHAEL & MAZZA, EDOARDO 2007 Electromechanical coupling in dielectric elastomer actuators. *Sensors and Actuators A: Physical* **138** (2), 384–393.
- WU, CHUNLIN, DENG, JIANSONG & CHEN, FALAI 2007 Fast data extrapolating. *J. Comput. Appl. Math.* **206** (1), 146–157.
- YANG, JIANMING & BALARAS, ELIAS 2006 An embedded-boundary formulation for large-eddy simulation of turbulent flows interacting with moving boundaries. *J. Comput. Phys.* **215** (1), 12–40.
- YU, ML, WANG, ZJ & HU, HUI 2011 A high-order spectral difference method for unstructured dynamic grids. *Computers & Fluids* **48** (1), 84–97.
- YU, MEILIN, WANG, Z. J. & LIU, YEN 2014 On the accuracy and efficiency of discontinuous galerkin, spectral difference and correction procedure via reconstruction methods. *J. Comput. Phys.* **259**, 70–95.
- ZHANG, QINGHAI 2011 High-order, multidimensional, and conservative coarse-fine interpolation for adaptive mesh refinement. *Comput. Methods Appl. Mech. Eng.* **200** (45-46), 3159–3168.
- ZHANG, QINGHAI, JOHANSEN, HANS & COLELLA, PHILLIP 2012 A Fourth-Order Accurate Finite-Volume Method with Structured Adaptive Mesh Refinement for Solving the Advection-Diffusion Equation. *SIAM Journal on Scientific Computing* **34** (2), B179+.

- ZHANG, W, HOWELL, L, ALMGREN, A, BURROWS, A & BELL, J 2011*a* CASTRO: A new compressible astrophysical solver. II. Gray radiation hydrodynamics. *The Astrophysical Journal Supplement Series* **196** (2), 20.
- ZHANG, W, HOWELL, L, ALMGREN, A, BURROWS, A, DOLENCE, J & BELL, J 2013 CASTRO: A new compressible astrophysical solver. III. Multigroup radiation hydrodynamics. *The Astrophysical Journal Supplement Series* **204** (1), 7.
- ZHANG, ZHI-QIAN, YAO, JIANYAO & LIU, GR 2011*b* An immersed smoothed finite element method for fluid–structure interaction problems. *International Journal of Computational Methods* **8** (04), 747–757.
- ZHAO, XUANHE, HONG, WEI & SUO, ZHIGANG 2007 Electromechanical hysteresis and coexistent states in dielectric elastomers. *Physical review B* **76** (13), 134113.

Appendix B. Users Guide (appended pdf document)

Flow Code Design Document

M. Jemison

HPC Engineering Group
Florida State University
Florida A&M University

April 22, 2016

Contents

Table of Contents	ii
List of Figures	iii
1 Installation and FAQ's	1
1.1 Installation Guide	1
1.2 Version Notes	2
1.3 Frequently Asked Questions	3
2 Interpolation & Differencing Operators	7
2.1 Interpolation Operators	7
2.2 Differencing Operators	8
2.3 Data Synchronization	10
3 Practices	13
3.1 Filling Ghost Values	13
3.2 Convergence	15
3.3 Low Level Functions	15
3.4 Linear Solver	15
3.5 Operator Composition	15
3.6 Regridding	16
4 Implementation of Morinishi (2010) Algorithm	17
4.1 Implementation Issues	17
4.2 Conservation Equation	17
4.3 Tensor Stress Terms	18
4.4 Time Integrator	20
4.5 Sponge Boundary Conditions	20
4.6 Code Structure	22

5	Interface Implementation	25
5.1	Interface Detection	26
5.2	Boundary Extrapolation Method (BEM)	28
5.3	Implementation	30
5.4	NACA-00XX Level-Set Function	38
6	MPI Details	43
6.1	Function Hooks	43
6.2	MPI Interface	44
6.3	MPI FAQ's	47

List of Figures

1.1	Recommended file structure: in some master director install BoxLib, LAPACK (if not already installed), open-mpi (if not already installed), and the energy code each in its own directory. For linking purposes, it may be easier on some systems to install a new local instance of MPI.	1
3.1	$\mathcal{L}^\infty$ errors for the cell-based (left) and face-based methods.	16
4.1	Error for passive advection of a scalar on a periodic domain, for 3 levels of AMR with base resolution 32^2 . Error shown for RK2, CFL=0.5 (blue), RK3,CFL=0.5 (red), and RK3,CFL=0.25 (green).	19
4.2	Logical map for the compressible flow code.	22
5.1	Error in zero level-curve computation of the star function.	26
5.2	Zero level-curve of the star function. Cut cells are denoted with red circles. Location of and orientation to the interface are shown in red. Resolution is $N=64^2$	27
5.3	Trailing edge of an airfoil without (left) and with (right) the root-finding correction in (5.7). The blue line shows the exact interface for a NACA0012 airfoil with a 15° angle of attack, and the red circles show the computed roots of the level-set function. Chord length is $C = 1$, with mesh spacing $\Delta x = C/100$. The roots are incorrectly predicted by the naive formula (5.6) when the trailing edge is under-resolved by the mesh, but accurately captured using (5.7).	28
5.4	Projection error of function (5.20) to the star-shaped interface.	31

5.5	A solid structure (shaded) cutting an adaptive grid. Cut cells (circles) use fine values (filled) or coarse values (open) for the least-squares reconstruction, depending on their proximity to the coarse/fine mesh boundary. Only cells with the surrounding 7×7 stencil in the fine grid use fine grid cells for the reconstruction. Ghost cells are outlined with dotted lines.	33
5.6	A solid structure (shaded) cutting an adaptive grid. Cut grid cells (filled circles) have a neighboring cell in the liquid (white). Ghost cells adjacent to the fine grid are outlined with dotted lines. Fine level cells on the coarse/fine boundary (open circles) require ghost values filled from the underlying coarse grid.	36
5.7	A map of the functions written for the interaction of the flow with the interface. Arrows point from the caller to the callee. The fluid algorithm <i>only</i> calls <code>ml_bem_recon_all</code> , which calls the rest of the necessary functions. Reconstructions for <i>all</i> state variables on <i>all relevant</i> levels are performed from here.	37
5.8	Flow chart for the execution of the solid wrapper function <code>ml_bpm_recon_all</code>	38
5.9	Illustration of the objective function, i.e. the magnitude of vector $\mathbf{e}$	39

Chapter 1

Installation and FAQ's

1.1 Installation Guide

Necessary items for installation of CODE

- Lapack (tested version `lapack-3.5.0`)
- MPI package (using `OpenMPI`)
- BoxLib package (distribution version)
- `gfortran` compiler

In the BoxLib User's Guide (distributed in the package), it says that it is sufficient to include the path to Boxlib in the *local* directory. However, compilation errors have been found to occur. It is advisable to set a `BOXLIB_HOME` variable in your `.bash_profile`. Additionally, set variables `MPHOME` and `LAPACK` to correctly compile and link the necessary libraries. These instructions are given under the assumption that you *do not* have root access. If you do, you can simply install executables in the `/usr/bin` directory. If not, I recommend that you install the MPI and Lapack libraries/executables in their own directories. This will make linking as painless as possible.

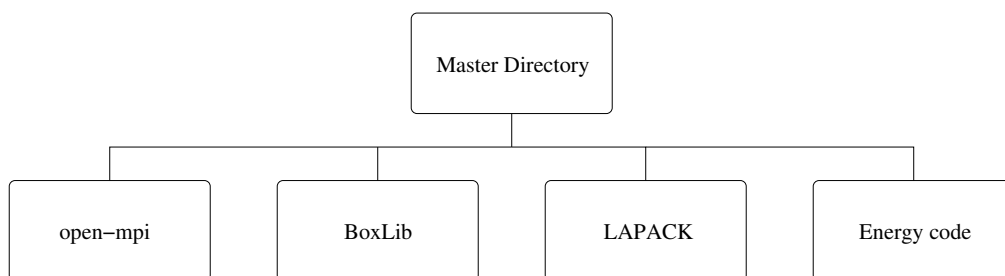


Figure 1.1: Recommended file structure: in some master director install BoxLib, LAPACK (if not already installed), open-mpi (if not already installed), and the energy code each in its own directory. For linking purposes, it may be easier on some systems to install a new local instance of MPI.

1. Add `BOXLIB_HOME`, `MPIHOME`, and `LAPACK` variables declared in your `.bash_profile`
2. Compile Lapack using the correct `make.inc.*` configuration for your machine. This must be installed first as several routines in the code require the linear solvers. Using the CMAKE configure utility can make this easy, otherwise you must individually change each `Makefile` to use the correct `make.inc.*` file. Typing “make” in the main directory will toss errors for each makefile that needs to be changed and point you to the appropriate directory.
Best practice: Execute a `make cleanall` to ensure a clean build environment, `make blaslib`, then use `make all`.
3. Configure and compile the MPI library for your Fortran compiler. If an MPI executable is installed in the directory `/usr/local/bin/`, the environment variable should be set to `export MPIHOME=/usr/local`. If you do not have root access, then it is recommended that you use the configuration tool, specifying to installation in the MPI directory.

```
./configure --prefix=$MPIHOME
make all install
```

The executables and libraries will be found in `/bin` and `/lib` subdirectories respectively. The BoxLib files will automatically link to `$MPIHOME/lib`

4. Compile the energy source code files.
 - You will likely need to make changes to the `GNUMakefile`. All files specific to the energy code are distributed in the `/Energy_code` directory. Check the `VPATH` file inclusions in the makefile for your machine and where you put all of your files.
 - If the error `library not found for -ltmglib` occurs during compilation, this indicates that Lapack was not properly installed or linked. Check the file `BoxLib/Tools/F_mk/GMakedefs.mak`. You may not be specifying the correct location or name of the libraries.
 - Note: The BoxLib source files will be compiled during this stage. There is no need to separately compile the BoxLib library.
5. An executable for your system should be created in the directory. You can run a sample test for flow past a cylinder (FPAC) by executing `./main.ARCH.gfortran.MPI.exe inputs_fpac_2d`.

1.2 Version Notes

There have been several key changes made to the BoxLib source that are necessary for full functionality with the code. I have off-set these changes in the BoxLib source distributed with the code with a note, to the effect of **Changed**, **Jemison DATE**. Changes include, but are not limited to,

- `make_new_grids.f90`: Changed `ng_buffer` to 8, to allow for 4th order ghost fill. This also helps eliminate some detrimental coarse/fine grid interactions.
- `parallel.f90`: Added function hooks to call MPI subroutines to send and receive a single integer or logical. Only hooks for multidimensional arrays were included. This prevents the user from having to type the full MPI send/receive commands each time, wrapping the more complicated call in a simpler package. These hooks include

- `parallel_send_i`: Send a single integer. Useful for passing sizes for allocatable arrays.
- `parallel_recv_i`: Receive a single integer.
- `parallel_send_l`: Send a single logical. Useful for communicating whether or not processors are finished with a portion of code.
- `parallel_recv_l`: Receive a single logical.

Note that whether you are running the code with MPI or not, it should be compiled with the MPI flag set to ‘true.’ The above hooks require “stubs” (i.e. trivial calls with an appropriate return value) in the file `parallel_stubs.f90`. As of April 22, 2016, this has not been added, so compilation will not be successful.

1.3 Frequently Asked Questions

I am getting a “malformed archive” error when compiling one of the libraries. What’s happening?

Try doing a `make clean` and then `make` to recompile everything from scratch. Also, check your `$MPIHOME` and `$LAPACK` profile variables. They should point to the directory containing `bin` and `lib` subdirectories or the `*.a` archive files. It is possible that you are pointing to some other directory where the archive cannot be properly built.

When I compile the energy code from scratch, compilation proceeds up to `parallel.f90`, then I receive the error “Can’t find `mpif.h`.” How do I solve this?

This indicates a linking error, in that the compiler can’t find the MPI library. The easiest way to solve this is to alter the file `Boxlib/Tools/F_mk/GMakedefs.mk`. For a GNU-based MPI configuration, use

```
mpi_libraries = -L$(MPIHOME)/lib -lmpi_mpfh
mpi_include_dir = $(MPIHOME)/include
mpi_lib_dir = $(MPIHOME)/lib
```

For a Cray-based MPI configuration, use

```
mpi_libraries = -L$(MPIHOME)/lib -lmpichf90
mpi_include_dir = $(MPIHOME)/include
mpi_lib_dir = $(MPIHOME)/lib
```

I have installed OpenMPI, but I can’t just use MPI commands on the command line. Why?

In your root directory, open `~/.bash_profile` and add `$(MPIHOME)/bin` to the path variable. This tells your command line where to look for the binaries.

I have compiled OpenMPI, LAPACK, and the energy code, but it fails in the last block, with the error /usr/bin/ld: cannot find -lblas. What do I do?

The last step is the linker. It links the compiled files into a single executable. The easiest way to solve this is to alter the file `Boxlib/Tools/F_mk/GMakedefs.mk`.

```
lapack_libraries = -L/$(LAPACK) -llapack -lrefblas
```

On different systems, LAPACK creates archive libraries called `libblas.a` or `librefblas.a`. You have to look at your system and link the proper archive.

When I compile LAPACK, I get a “malformed archive” error for librefblas.a. What do I do?

Type `make blaslib` to make the library. Then, type `make` to continue compiling as normally.

Why can't I link the LAPACK libraries?

When you compile the libraries, you create an archive called `liblapack` and `libblas` or `librefblas`. To properly link these libraries, you remove the “ib” in the library, so `liblapack` becomes `llapack` and `libblas` becomes `lblas` (as implemented in the `GMakedefs.mk` file). On some systems, the BLAS (Basic Linear Algebra Subroutine) library compiles to `librefblas`, so you may need to change the linking in `/BoxLib/Tools/F_mk/GMakedefs.mk`. Look for the variable `lapack_libraries` and make the appropriate changes.

When I compile OpenMPI, it gives me an error about dynamic/static libraries. How do I fix this?

This is a system-dependent error. This has been encountered on DOD machines. It is likely due to the type of compilers natively used by the system. Try explicitly specifying the GNU compilers, so use `CC=gcc`, `CXX=g++`, and `FC=gfortran`. Other modifications to the environment variables may be necessary. Contact your friendly neighborhood sysadmin if problems persist.

Why can't I compile without MPI? I'm getting an error about some MPI functions.

Not your fault. I had to write a few MPI hooks for my interface subroutines, which require a “stub” caller (that doesn't actually do anything) to work without MPI. See Sec. 1.2. That's on the to-do list.

I'm on a Linux box, and I'm getting a segmentation fault when I try to print out mesh stats at the end of the main body of the code. Why?

This code was developed and tested primarily on a Mac. You can disable the call to print information. It doesn't appear to affect the accuracy of the code during a run. You can also try

running on a different machine. Other than that, your guess is as good as mine. It's on the to-do list.

Why am I seeing weird errors when I try to move the solid level-set when using MPI?

Running the diagnostic tests will show that each processor initializes its own `probin.f90` that initializes the testing parameters. Each processor has its own copy of the parameters that doesn't know about the others. So, if one processor changes the level-set on its own, the others won't know. All processors need to change the level-set parameters (`xxblob`, `yyblob`, and `radblob`) to be the same. Include parallel barriers to make sure that values are synchronized before moving on.

I want to make a movie from my results. How do I do that?

First, download VisIt visualization software. It's free software from Lawrence Livermore. Then, just type `ls -l plt*/Header | tee movie.visit` in your terminal. Then, open the movie file in VisIt.

Why are my arms so small? I can't type with these fuzzy paws.

You may be a koala. Please contact your systems administrator.

Chapter 2

Interpolation & Differencing Operators

2.1 Interpolation Operators

There are currently 5 interpolation modules. Interpolation modules may contain a number of specialized functions for computing product interpolations.

- `interpolate_time`
- `interpolate_to_face`
- `interpolate_to_cell`
- `interpolate_to_cc`
- `interpolate_to_node`

Each module has further subroutines to interpolate cell- or face-centered data to the desired location.

2.1.1 Module: `interpolate_to_face`

Accepts a cell-centered `MultiFab` with an arbitrary number of components (`ncomp`) and returns a face-centered `MultiFab` with `ncomp` components for each of `dim` face directions. This is used to calculate the conservative and non-conservative advective terms in the momentum equation, where each directional advective term must be interpolated in each direction.

2.1.2 Module: `interpolate_uucc_to_face`

Accepts a cell-centered `MultiFab` with `dim` components (`ncomp`) and returns a face-centered `MultiFab` with 1 component in each of `dim` face directions, with the i^{th} component interpolated in the i^{th} direction. This is used to interpolate a cell-centered velocity to a face-centered velocity. While this functionality duplicates that of `interpolate_to_face`, it is useful for calculating momentum or divergence terms.

2.1.3 Function: `interpolate_to_face_prod` (deprecated)

Accepts cell-centered variables $\mathbf{g}_{cell}$ and ∇u_{cell} , i.e. $\bar{g}_i^{x_i}$ and $\delta_{x_i} u_i$, suppressing Einstein summation. Input is assumed to have DIM components. Performs direct component-by-component multiplication, and then takes the average in the “major” direction. Output will be a face centered multiFAB with one component in each of DIM directions. This is to be used for the non-conservative momentum equation ONLY!

$$\frac{\mathbf{g}_{i+1} \nabla \mathbf{u}_{i+1} + \mathbf{g}_i \nabla \mathbf{u}_i}{2} \quad (2.1)$$

2.1.4 Function: `interpolate_node_to_face_prod` (deprecated)

Accepts nodal variables $\mathbf{g}_{node}$ and ∇u_{node} with DIM components, i.e. $\bar{g}_i^{x_j}$ and $\delta_{x_j} u_i$, for $i \neq j$. Performs “cross” component-by-component multiplication, and then takes the average in the x_j direction. Output will be a face centered MultiFab with one component in each of DIM directions. This is to be used for the non-conservative momentum equation ONLY!

2.1.5 Module: `interpolate_to_cell` (deprecated)

Accepts a *single* component face-centered multifab with multiple dimensions. The input must have only one component, or the code will go into exit failure mode. The face-centered values are interpolated to cell-centered values. The output will be a cell-centered multifab with DIM components. Each component will have the form $\bar{u}_i^{x_i}$. This functionality has been absorbed into the computation of advective terms.

2.2 Differencing Operators

Differencing operators are located in the files `calc.visc_exp` and `coll.calc_flux`. This file also includes routines for calculating the stress tensor and the divergence of the stress tensor.

2.2.1 Function: `grad_cell_ff`

Accepts a cell-centered variable and calculate the face-centered gradient of each of an arbitrary number of components. This can be used for energy or velocity. Ordering of the output for multiple components is as follows for each face direction.

1. $du/dx, dv/dx, dw/dx$
2. $du/dy, dv/dy, dw/dy$
3. $du/dz, dv/dz, dw/dz$

This function is used to compute the non-conservative advective terms for the momentum and energy equations.

2.2.2 Function: grad_cell_cell

This function uses a 3-point centered difference stencil to compute a first order approximation to a cell-centered gradient, using cell-centered values. The **MultiFab** input can have an arbitrary number of components. Output is ordered such that ϕ_1 is differentiated with respect to all directions first, then ϕ_2 is differentiated with respect to all directions second, etc.

This is used to compute the pressure gradient in the momentum equation. In the energy equation, this is used to compute the tensor terms in the viscous heating term, $\tau_{ij} \frac{\partial u_i}{\partial x_j}$.

2.2.3 Function: grad_face_cc (deprecated)

Accepts a face-centered variable and calculate the cell-centered gradient of each of the facial components in the “principle” direction, i.e. $\frac{du}{dx}, \frac{dv}{dy}, \frac{dw}{dz}$.

Update: The functionality of this has been absorbed into the advective term functions. This may be either re-used or re-written for the purposes of refluxing.

2.2.4 Function: grad_face_cc_prod (deprecated)

Given two face-centered inputs, performs component-by-component multiplication and then takes the gradient. This will result in a cell-centered output with DIM components. This should be used with the conservative energy flux formulation. This may work, unaltered, after the face-centered energy is calculated.

2.2.5 Function: grad_cell_ff_moflux (deprecated)

Accepts two cell-centered variables, each with DIM components, and calculate the face-centered gradient of the component-by-component product of these variables. This is to be used specifically with cell-centered momentum and velocity. A face-centered quantity with 1 component in each direction is produced on output.

$$\begin{pmatrix} \frac{\partial}{\partial x} (\overline{g_1^x u_1^x}) \\ \frac{\partial}{\partial y} (\overline{g_2^y u_2^y}) \end{pmatrix}$$

2.2.6 Function: grad_face_node (deprecated)

Takes the gradient of the components of the face-centered variable in the direction orthogonal to the face-direction. Results in a nodal gradient, with DIM*(DIM-1) components. In 2D, the following data structure is produced on output.

$$\begin{pmatrix} \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} \end{pmatrix}$$

Update: Nodal values are no longer used in the collocated code.

2.2.7 Function: `grad_node_face_prod` (deprecated)

Accepts two pieces of node-centered data ($\overline{g_j^{x_i}}$ and $\overline{u_i^{x_j}}$) with DIM components. Outputs the gradient of the nodal data on the faces. The ordering is unintuitive. Differencing nodal data in the y-direction along the x-face will give x-face data.

$$\begin{pmatrix} \frac{\partial}{\partial y}(\overline{g_2^x \overline{u}^y}) \\ \frac{\partial}{\partial x}(\overline{g_1^y \overline{v}^x}) \end{pmatrix}$$

Update: Nodal values are no longer used in the collocated code.

2.3 Data Synchronization

BoxLib includes a number of routines for synchronizing “ghost” data, i.e. data outside of a given multiFAB. This ghost data may be used to enforce boundary conditions or compute fluxes at a coarse-fine boundary. There are several cases for ghost data, which require that different synchronization routines are called.

1. Ghost data between FAB’s at the same mesh resolution.
2. Ghost data outside of the problem domain.
3. Ghost data at a coarse-fine boundary.

2.3.1 Type 1 Ghost Data: Same Mesh Level

Ghost data (whether cell-centered or face-centered) between FAB’s of the same mesh resolution is the simplest case. Data is simply injected from one grid to another. No interpolation or calculation is necessary, as all ghost data locations correspond to true grid data locations for another FAB. This data can be synchronized using the command `call multifab_fill_boundary(data(ilev))`. This has been tested for faces using the face-to-nodal interpolation subroutine, which requires ghost values on cell faces. This has not been tested for nodal data, as the program currently does not use any ghost nodal values.

2.3.2 Type 2 Ghost Data: External Boundary Data

Data outside of the problem domain is typically used to enforce boundary conditions. I have been told by one of the LBNL scientists responsible for BoxLib that ghost data outside of a wall-like boundary corresponds to data *on the wall*. Wall-like boundaries would include any boundary with homogeneous Neumann boundary conditions, e.g. inflow, no-slip, free-slip. How this data is synchronized is dependent on the location of the data and the boundary condition. Periodic boundary conditions are viewed as an *internal* boundary, and is handled using a periodic masking of data. For cell-centered and face-centered data, this is applied again using the command `call multifab_fill_boundary(data(ilev))`. This has again been tested using the face-to-nodal interpolation operator for face-centered data, and it gives the proper second order convergence in the $\mathcal{L}_{avg}^1$ and $\mathcal{L}^\infty$ error norms on a single grid.

LBLN’s research codes appear to use *co-located* data, i.e. all data is in the same place. They put all variables natively at cell-centers, so cell-centered data is handled with default routines. Physical

boundaries applied with `multifab_physbc(data(ilev),startcomp,bcstart,numcomp,bc(ilev))`. See the file `interpolate_time.f90` for an example. Variable `startcomp` refers to the component in the multiFAB at which to start applying boundary conditions. Variable `bcstart` is the component in the boundary tower array at which to find boundary conditions for the given state variable. Note: the boundary tower will hold BC's for velocity and scalars, so pay attention to how this is applied. Variable `numcomp` is the number of components to which boundary conditions are applied. Variable `bc(ilev)` refers to the object `the_bc_tower%bc_tower_array(ilev)`, which contains information about boundary conditions for advection and the linear solver.

2.3.3 Type 3 Ghost Data: Coarse-Fine Boundary Data

Coarse-fine data requires interpolation, as fine level data will not be located at the same physical point as coarse level data. Cell-centered data is synchronized using the function `multifab_fill_ghost_cells`. From exploring the code, it appears that `BoxLib` uses coarse cell values to compute a centered-difference approximation to the cell gradient at the coarse cell which holds the fine cells in question (requiring at least one coarse ghost cell). The coarse gradient approximation is used with the offset of fine data from the coarse data to approximate fine level data. This approximation appears to degrade accuracy for the gradient at the coarse-fine boundary, as it uses non-local data to approximate a fine level cell.

`BoxLib` has a pre-packaged 4th order ghost fill routine for cell-centered data. Tests for stationary problems seem to indicate that it is conservative. This can be triggered by setting the optional variable `fourth_order_input=true` when calling `multifab_fill_ghost_cells`. It has been found to produce second order convergence in the presence of AMR for interpolation, gradient, and diffusion operators. There is not implementation of this operator in 3D. This will either need to be developed, or the errors associated with the second-order ghost fill operator will have to be accepted.

2.3.4 Current Version Notes

The current version fills all ghost data in the subroutine `advance_all_rhs`, which calls subroutines to advance mass, momentum, and energy. Once density and energy are advanced, pressure is advanced using the ideal gas equation of state,

$$p = (\gamma - 1)\rho e_{int}. \quad (2.2)$$

Here, it is assumed that $\gamma = \frac{7}{5}$. In each of the subroutines called in `advance_all_rhs`, a 4th order ghost fill routine is called. These same subroutines also call a cell-centered restriction operator, to synchronize data between coarse and fine levels. The face-centered restriction operator is *only* called in the computation of the face-centered momentum. Because the “refluxing” is applied by this operation, the method appears to be mass conservative, up to machine precision, for the non-stationary tests that have been run thus far.

TODO: To apply refluxing for the conservative advective terms, several subroutines will likely need to be re-written. The conservative advection terms will need to be recalculated, so that only the face-centered product is computed. Re-fluxing (i.e. face-centered restriction) will then be applied, and finally the divergence of the face-centered flux can be computed. Similarly, in the momentum equation, the term $\frac{\partial \tau_{ij}}{\partial x_j}$ may need to be re-implemented such that face-centered terms are computed, re-fluxing is applied, and finally the divergence is taken. The energy diffusion terms should be broken up in similar fashion; after the face centered energy gradient is computed, re-fluxing should

be applied, and finally the divergence of the flux should be computed. It is not clear if there is a direct method for re-fluxing the viscous heating term, as it is not written as the divergence of a flux. Rather, it appears as a component of the divergence of a quantity under total energy conservation. Perhaps it will be gathered in with a Total Energy Conserving Re-Fluxing (TECReF) correction, along with the momentum viscous terms?

Chapter 3

Practices

3.1 Filling Ghost Values

At every grid level, we must fill cell-centered ghost values. It is currently assumed that edge data has no ghost cells, so no ghost values must be filled. “Good practice” would dictate that fine level data (assumed to be more accurate than coarse level data) should always be “restricted” (using Multigrid terminology) to coarser levels. This is done for cell- and edge-centered data. Data is operated on first by “levels” and then by FAB’s (Fortran Array Boxes). Indices dictating the bounding box of the FAB data are obtained first. A pointer is used to point to the data at the level and FAB we are operating on. The data in the pointer is sliced and passed into a function.

$$data(x, y, z, component)$$

A standard function will appear as follows.

```
do n = 1,nlevs
  do i = 1,nfabs(data(n))                ! number of Fortran Array Boxes
    ptr => dataptr(data(n),i)
    lo = lwb(get_box(data(n),i))
    hi = upb(get_box(data(n),i))
    select case(dim)
    case(2)
      call ccfuction(ptr(:, :, 1, :), lo, hi, numghost, dx(n, :)) ! 1 z-slice, all comps
    case(3)
      call ccfuction(ptr(:, :, :, :), lo, hi, numghost, dx(n, :)) ! all z-slices, all comps
    end select
  end do

  call multifab_fill_boundary(data(n)) ! fill ghost cells BETWEEN fabs at same level
  call multifab_physbc(data(n), 1, 1, numcomp, bc(n)) ! fill ghost cells at physical
                                                    ! boundary (wall, inflow, slip) only
end do
```

```

do n=nlevs,2,-1 ! restrict, starting at finest and working down
  ! Synchronizes coarse data underlying a fine mesh.
  call ml_cc_restriction(data(n-1),data(n),rr(n-1))

  ! Fills coarse/fine ghost cells
  call multifab_fill_ghost_cells(data(n),data(n-1),numghost,rr(n-1,:), &
    bc(n-1),bc(n),1,1,numcomp)
end do

```

If a test works for a single level, but does not work for multiple levels, this is an indicator that your ghost cell values. A good practice is to initialize ghost values to an absurd value. In Fortran, a good value may be $-9.999e2$. This should cause things to go wrong quickly in testing, which is what you want. If you are going to fail, fail spectacularly. Ghost values are overwritten at fine-fine mesh boundaries (i.e. between FAB's at the same level) using fine level data. Ghost values at the fine-coarse boundary are overwritten using interpolated data from the coarse level to the fine level. Regardless, the values should ALWAYS be overwritten from the default before use.

3.1.1 Function: multifab_fill_ghost_cells

The function `multifab_fill_ghost_cells` uses coarse cell values to fill fine level ghost values. This is done in contrast to using low- or high-order interpolation to extrapolate the solution on the fine level mesh to fine level ghost cells. While nothing is wrong with this, it is implemented in a fashion that exhibits poor behavior for even moderate gradients on multilevel meshes. Slope limiting is used by default, but an unlimited slope is used (i.e. it is not enforced that the reconstructed slope not produce any additional extrema) for this application. A second-order centered difference operator is used to define the gradient in the x- and y-directions, centered in the coarse level cell that holds the fine level ghost cells. An “off-set” from the coarse cell-center to the fine ghost cell centers is computed. The fine level ghost values is then computed as

$$fine_{i,j} = coarse_{2i,2j} + \text{off}_x \frac{coarse_{2(i+1),2j} - coarse_{2(i-1),2j}}{2} + \text{off}_y \frac{coarse_{2i,2(j+1)} - coarse_{2i,2(j-1)}}{2} \quad (3.1)$$

Note: The factor of Δx_c or Δy_c that would appear in the numerator of the offset and the denominator of the centered difference is omitted, as it will cancel in multiplication (in the application, the routine which performs this operation does not have access to Δx_c).

While this method is formally second-order accurate, it is observed in practice (even on benign test problems), to produce poor results when computing differences across the coarse-fine boundary. This operation, which occurs within the subroutine `fillpatch→interp`, could be altered via a number of strategies. It should be noted that the centered difference data is not “local” relative to the fine level ghost cells. This operation will use values that are distance $> 3\Delta x_f$ away from the point of interest. A more local (and also second order accurate) implementation would use the four cell-centered values that make up the vertices of the nodal control region. The ghost values in four different coarse level cells would be filled at each “node” on the mesh. This may have implications as to data locality and parallelization, but it will have the same stencil width as the currently implemented operator (i.e. it requires only one coarse level ghost cell). Additional benefits include the fact that the interpolation error for the proposed operator is proportional to $|\sum_i \frac{\partial^2 f}{\partial x_i^2}|$, rather

than $|\sum_i \frac{\partial^3 f}{\partial x_i^3}|$, the error of the currently implemented method. Look at modifying this based on the variable `CC_CROSS_STENCIL`.

3.2 Convergence

Note that using “restriction,” i.e. using fine level data to overwrite coarse level data, will apparently cause slower convergence. This is because the more fine level data “pollutes” the coarse data with more accurate values. The coarse level data will become more accurate, but the difference between the coarse and fine level data will be less pronounced. The best way to verify a module is to write it without restriction and test that first. After adding a restriction step to a second order operator, expect a convergence rate of approximately 1.5 when using AMR.

Also, note that fine level cells that utilize ghost data will experience degradation in accuracy, as fine-level ghost cells are filled using underlying coarse level data.

3.3 Low Level Functions

Low level functions operate on a single FAB. Input is typically a pointer to some slice of a multiFAB object. The input arguments in FORTRAN are re-indexed using the `lo` and `hi` FAB bounds for uniformity across platforms (Fortran natively starts indices at 1, while C++ starts from 0). An offset may be used to account for ghost cells, but this can **EASILY CAUSE PROBLEMS**. If you re-index using more ghost cells than your FAB has access to, this will cause seemingly random errors that pollute your data. It may cause overflow, a segfault, or a very subtle error, depending on what is stored in the out of bounds index you are attempting to access. If you are lucky, your code will always crash. Protip: Run the code several times on the same problem and examine some fixed error metric. If the error changes for sequential runs of the same test case, you may have this problem.

3.4 Linear Solver

This code uses an adapted linear solver from the BoxLib testing suite, taken from the directory `BoxLib/Tutorials/HeatEquation_EX5_F/advance.f90`. A later version of the code will pass in a copy of the scalars, as density, enthalpy, and momentum will be necessary to setup the coefficients and right-hand side of the linear problem. Note: The solution variable passed in is used as the initial guess, and the solution is returned in place. You *MUST* use `call multifab_fill.boundary` on the solution variable before it is passed into the linear solver. If you don’t, the level ghost cells will not be properly filled between FAB’s at the same level. This will result in a segfault that seems to result from working code. DO NOT forget this. TODO: move fill boundary call into the routine to make sure that it is done properly?

3.5 Operator Composition

While it is tempting to implement each interpolation and differencing operator individually and then compound them, this is observed to lead to large degradation in accuracy. Consider the Laplacian

operator for a face-centered variable u :

$$u_t = \delta_{x_i} \cdot (\nu \delta_{x_i} u). \quad (3.2)$$

A naive implementation would difference the face values to cells, fill ghost cell values, and then difference the cell values to faces. The results in severe loss of accuracy. The problem is that the error in the differencing is compounded to the interpolation, resulting in much lower accuracy. This effect is amplified for non-linear terms, such as conservative momentum flux. Figure 3.1 shows the difference in MAX-norm error behavior for the cell-based vs. the face-based methods. A degradation to first order long-term accuracy is observed when using the cell-based method. I conclude that this method is not to be used, and that when an operator would require a call to a ghost filling routine, it should be written as a single operator.

$$\delta_t u_i + \delta_{x_j} (\overline{\rho^{x_j}} u_j^{x_i} \overline{u_i^{x_j}}) = \dots \quad (3.3)$$

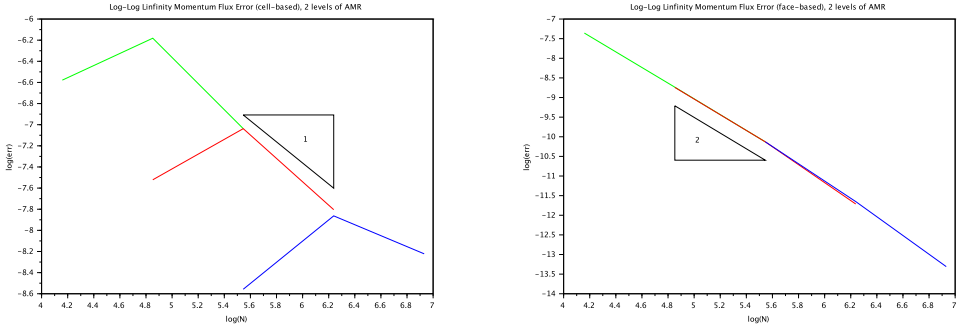


Figure 3.1: $\mathcal{L}^\infty$ errors for the cell-based (left) and face-based methods.

3.6 Regridding

IMPORTANT: The `BoxLib` Fortran source file `make_new_grids` has been changed. The `BoxLib` design document has an inaccuracy in it, regarding the nesting of grids. They claim that between level Ω^l (coarse) and level Ω^{l+1} (fine), there are 4 level Ω^l “buffer” cells. This seems to be incorrect. When running the “Random Wave” test with Laplacian-based refinement criteria, situations arise that result in a buffer of only 2 level Ω^l cells. I believe that they meant that there are 4 level Ω^{l+1} cells. We need 8 $l+1$ cells. This has been corrected by making two changes.

1. File `regrid.f90`: Set the variable `ng_buffer = 8`. The `regrid` function lives in the `varden` directory and is dependent on the overall code.
2. File `make_new_grids.f90`: Set the variable `ng_buffer = 8` in function `enforce_proper_nesting`. Note that this file is a GLOBAL file. If the `BoxLib` repository is updated, this change will be overwritten. You need to *BE VERY CAREFUL* about this.

Chapter 4

Implementation of Morinishi (2010) Algorithm

The Morinishi (2010) algorithm for staggered grids is presented here. The methods are collected in a module `morinishi_advance`, which calls subroutines that compute the necessary quantities. The time integrator is currently in this routine. Variables will need to be updated in lockstep with one another.

4.1 Implementation Issues

I know for a fact that the native 4th order ghost fill routine is unstable. It crashes immediately upon attempting to fill if `ng_cell > 2`. This is problematic as it dictates the type of stencils that may be used at coarse/fine mesh boundaries.

The routine `morinishi_advance_density` calls `multifab_fill_boundary` to ensure that intra-level ghost cells are properly filled. No ghost filling is applied in `morinishi_calc_momentum` because we never use ghost momentum values. Momentum is used only to calculate the divergence of momentum in the routine to advance density. For the second order Runge-Kutta method, the intra-level ghost values are filled in `morinishi_advance_density` for both `sc_newk` and `sc_newkp1`. A cell-centered restriction and coarse/fine ghost fill are performed at the end of `morinishi_advance_density`. So, upon completion of the advancement, all ghost cells should be filled.

4.2 Conservation Equation

The module `morinishi_advance` computes a face-centered momentum. This only has the “major” components (i.e. $\bar{u}^x, \bar{v}^y$). This is computed in the main function, as it will be needed for the momentum flux calculation as well. A secondary subroutine `div_momentum` takes the divergence of

this quantity. A simple RK2 algorithm is applied.

$$\begin{aligned}
y_n &= \rho^n \\
f(t_n, y_n) &= -\nabla \cdot \mathbf{g}^n \\
y_{n+1/2} &= y_n + \frac{\Delta t}{2} f(t_n, y_n) \\
y_{n+1} &= y_n + \Delta t f\left(t_n + \frac{\Delta t}{2}, y_n + \frac{\Delta t}{2} f(t_n, y_n)\right)
\end{aligned}$$

This, with the second order spatial discretization, has been verified to be second order accurate for passive advection. However, problems with the second order discretization have been found for multiple levels of AMR. For long term time integration, the appearance of high-frequency $2\Delta x$ modes in the wake of the flow destroys the long-term accuracy. For this reason, a third-order, three-stage Runge-Kutta method has been implemented. This appears to reduce errors that are likely associated with dispersion of the numerical scheme. The RK3 scheme is presented here for the continuity and momentum equations.

$$\delta_t \rho = -\nabla \cdot \mathbf{g} \equiv k(\rho, \mathbf{u}) \quad (4.1)$$

$$\delta_t (\rho u_i) = \delta_{x_j} \tau_{ij} - \delta_{x_i} p - \frac{1}{2} (\delta_{x_j} (g_j u_i) + \rho u_j \delta_{x_j} u_i) \equiv h(\rho, \mathbf{u}, p) \quad (4.2)$$

$$k^n = k(\rho^n, \mathbf{u}^n), \quad h^n = h(\rho^n, \mathbf{u}^n, p^n) \quad (4.3)$$

$$\rho^{n+1/3} = \rho^n + \frac{\Delta t}{3} k^n, \quad u_i^{n+1/3} = \frac{\sqrt{\rho^n} u_i^n + \frac{\Delta t}{3\sqrt{\rho}} h^n}{\sqrt{\rho^{n+1/3}}} \quad (4.4)$$

$$k^{n+1/3} = k(\rho^{n+1/3}, \mathbf{u}^{n+1/3}), \quad h^{n+1/3} = h(\rho^{n+1/3}, \mathbf{u}^{n+1/3}, p^{n+1/3}) \quad (4.5)$$

$$\rho^{n+2/3} = \rho^n + \frac{2\Delta t}{3} k^{n+1/3}, \quad u_i^{n+2/3} = \frac{\sqrt{\rho^n} u_i^n + \frac{2\Delta t}{3\sqrt{\rho}} h^{n+1/3}}{\sqrt{\rho^{n+2/3}}} \quad (4.6)$$

$$k^{n+2/3} = k(\rho^{n+2/3}, \mathbf{u}^{n+2/3}), \quad h^{n+2/3} = h(\rho^{n+2/3}, \mathbf{u}^{n+2/3}, p^{n+2/3}) \quad (4.7)$$

$$\rho^{n+1} = \rho^n + \Delta t \frac{k^n + 3k^{n+2/3}}{4}, \quad u_i^{n+1} = \frac{\sqrt{\rho^n} u_i^n + \frac{\Delta t}{\sqrt{\rho}} \frac{h^n + 3h^{n+2/3}}{4}}{\sqrt{\rho^{n+1}}} \quad (4.8)$$

It is possible that the fourth-order routine that is used to fill ghost cells is causing this problem. The native routine in Boxlib is not necessarily conservative. It uses an averaging scheme, rather than a differencing scheme, to determine ghost cell values.

4.3 Tensor Stress Terms

The tensor stress terms are written in the standard fashion.

$$\tau_{ij} = \begin{cases} \mu (\delta_{x_j} (u_i) + \delta_{x_i} (u_j)), & i \neq j \\ \frac{4}{3} \mu \delta_{x_i} (u_i), & i = j \end{cases} \quad (4.9)$$

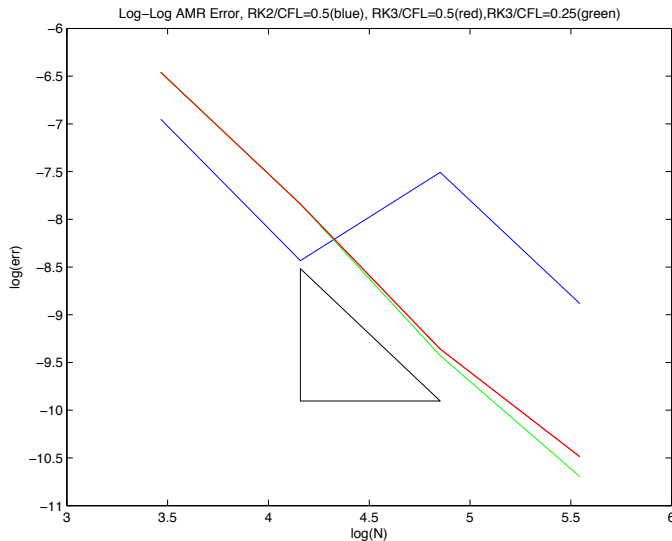


Figure 4.1: Error for passive advection of a scalar on a periodic domain, for 3 levels of AMR with base resolution 32^2 . Error shown for RK2, CFL=0.5 (blue), RK3, CFL=0.5 (red), and RK3, CFL=0.25 (green).

Tensor terms are discretized as in Wall, *et. al.* for a staggered grid scheme. For the momentum equation, in which the term

$$\frac{\partial}{\partial x_i} \tau_{ij} \quad (4.10)$$

is calculated, adjustments are needed for the collocated method.

The tensor term must be discretized in a fashion that preserves total energy. As such, individual tensor terms are all discretized at cell centers. Here, the operator δ_{2x_j} denotes the 3-point centered difference stencil at cell-centers.

$$\tau_{ij} = \begin{cases} \mu (\delta_{2x_j}(u_i) + \delta_{2x_i}(u_j)), & i \neq j \\ \frac{4}{3}\mu \delta_{2x_i}(u_i), & i = j \end{cases} \quad (4.11)$$

An equivalent form exists for the difference operator

$$\delta_{2x_i}(u) = \delta_{x_i}(\bar{u}^{x_i}) = \frac{u_{i+1} - u_{i-1}}{2\Delta x}. \quad (4.12)$$

Then, when discretizing (4.10), we again must use a centered difference stencil

$$\frac{\partial}{\partial x_i} \tau_{ij} \rightarrow \delta_{2x_i}(\tau_{ij}). \quad (4.13)$$

This results in a “diamond-shaped” stencil, with a width of five points, that induces a red-black decoupling. This could cause issues down the road. It is possible that other discretizations exist that

preserve total energy and do not exhibit such decoupling, but I was not able to construct any. The presence of dissipation due to the LES model (which should act solely to dissipate kinetic energy, while preserving the total amount of kinetic energy) may act in our favor here.

The tensor terms appearing in the energy equation are handled similarly. Note that τ_{ij} lives at cell-centers, so $\frac{\partial u_i}{\partial x_j}$ must be discretized at cell centers. As such, the 3-point centered difference stencil is used to discretize the velocity gradient.

$$\tau_{ij} \frac{\partial u_i}{\partial x_j} \rightarrow \tau_{ij} \delta_{2xj}(u_i) \quad (4.14)$$

4.4 Time Integrator

The Runge-Kutta time integrator has been consolidated in the function `morinishi_advance_comp`, which performs advection for compressible flow. The function `do_rk3_step` calculates the right-hand side terms for each of the mass, momentum, and energy equations. This is paired with the function `advance_all_rhs`, which accepts the old state values and newly computed right-hand side values from `do_rk3_step` to advance the simulation by a factor of Δt . This drastically simplifies the implementation of the algorithm at 4.1.

4.5 Sponge Boundary Conditions

To allow for inflow/outflow boundary conditions, the capability for a sponge boundary layer has been added. This is called after each stage of the Runge-Kutta time integrator (4 times per step for RK4). A “sponge” layer is a zone with strong diffusive (i.e. explicit filtering) properties. The filtering grows stronger in polynomial fashion toward the edge of the boundary, driving vortical structures to the mean flow. The sponge is present at both inflow and outflow. At the outflow, this prevents flow from entering the boundary and reduces the presence of reflected waves. At inflow, the sponge layer damps out reflected acoustic waves from the initial conditions. When these initial acoustic waves interact with the inflow boundary conditions (Dirichlet conditions), this causes severe ringing in pressure, which propagates throughout the domain. The presence of the sponge layer mitigates, but does not eliminate, these effects.

The sponge is implemented in the simplest filtering fashion, damping out the high-frequency $2\Delta x$ modes. Let $f_{i,j}$ denote some flow quantity and let $\tilde{f}_{i,j}$ denote the filtered quantity. Thus, the following explicit filtering formula is applied,

$$\tilde{f}_{i,j} = f_{i,j} - \sigma_s(x) \left[f_{i,j} - \frac{1}{4} (f_{i+1,j} + f_{i-1,j} + f_{i,j+1} + f_{i,j-1}) \right], \quad (4.15)$$

where

$$\sigma_s(x) = \alpha_s \left(\frac{x - x_{lo}}{x_{hi} - x_{lo}} \right)^{\beta_s}. \quad (4.16)$$

The values α_s and β_s are sponge coefficients, where $\alpha_s \in [0, 0.4]$ and $\beta_s \in [1, 2]$. The locations x_{hi} and x_{lo} denote the high and low boundaries of the sponge layer. An input flag in the `apply_sponge` function is used to denote the orientation of the σ_s function. If the `sp_outflow` flag is true, the function is increasing; if the flag is false, the function is decreasing. This allows the same function to work for both inflow and outflow sponge layers. The current implementation uses $\alpha_s = 0.4$ and

$\beta_s = 1.5$. The higher value of α_s adds stronger diffusive properties, which seems to prevent **BoxLib** from failing. The anecdotal wisdom is to allow at least 20 coarsest level cells between the end of the physical domain and the end of the sponge, which works well in practice.

Parameters that activate or deactivate sponge should be implemented in the input file rather than in the Fortran code. Additionally, the sponge strength and location should be specified in the input. This will allow for testing of parameters without the need for compilation for each test.

Note: There appears to be an intermittent error in **BoxLib** that occurs when multiple levels of mesh refinement coincide with the boundary. The error produced says that the box "lies outside of the domain." This is likely due to my use of a larger buffer region between levels of mesh refinement, which is required for fourth-order ghost cell interpolation. If this can be reliably duplicated, there is probably a parameter that prevents cells from being tagged for refinement within a certain distance of a physical boundary. This is as yet unresolved.

4.5.1 Testing

The sponge layer has been tested for the 50 vortex problem, with vortices initialized between $x \in [0.2, 0.7]$ and a background flow of $\mathbf{u} = (0.1, 0)$. The sponge is present at the inflow from $x \in [0, 0.15]$, which appears to damp most acoustic waves. The sponge at outflow is applied for $x \in [1, 1 + 20\Delta x]$. A periodic boundary has been applied to the North and South edges of the domain. These conditions have been tested for a single grid and for grids with up to 3 levels of adaptive meshes. Refinement is dissipated quickly as vortices move in the the outflow sponge region, which drastically speeds up execution.

4.5.2 Code Modification

Because **BoxLib** and the **warden** package were distributed with incompressible flow applications in mind, the boundary conditions are not appropriate for compressible flow, which requires ghost pressure values to be reasonable. The values in `inlet_bc.f90` have been changed to specify unitary density and pressure inflow. The variable `INLET_TRA` corresponds to internal energy at inflow, given an ideal gas assumption of $\gamma = 1.4$. If the boundary condition on pressure is set to `EXT_DIR`, then pressure in `multifab_physbc.f90` is set to the inflow pressure. This *is not* present in the standard **warden** implementation.

The file `define_bc_tower.f90` has been modified as well. A new boundary condition `SPEC_OUTFLOW` has been added. This uses a first-order extrapolation for all flow variables. The original outflow condition used an `EXT_DIR` condition for pressure. Use of the outflow condition was tested with a high-order extrapolation, but this produced severe spurious inflow without sponge. This can be re-tested with sponge, but first-order is likely sufficient. An additional `SPEC_INFLOW` boundary has been added, but is *not currently* used. This condition specifies unitary pressure value at inflow, whereas the original inflow condition, *which is currently used*, uses a first order pressure extrapolation. The extrapolation may be more stable in the presence of sponge, for reflected acoustic waves. More testing is required.

TODO: Only the low x-direction inflow conditions have been modified. All boundary conditions should be modified in order to make the code more robust. This of course needs to be implemented in 3D as well.

4.6 Code Structure

The fluid portion of the code is broken up into 22 subroutines, counting the wrapper/driver functions rather than the wrapper and lowest level code. Note that the computation of terms on the right-hand side of the momentum and energy equations share a number of functions. The functions `calc_adv_con` and `calc_adv_noncon` are usable with both scalar and vector valued functions. This allows for computation of velocity and energy advective terms by the same function.

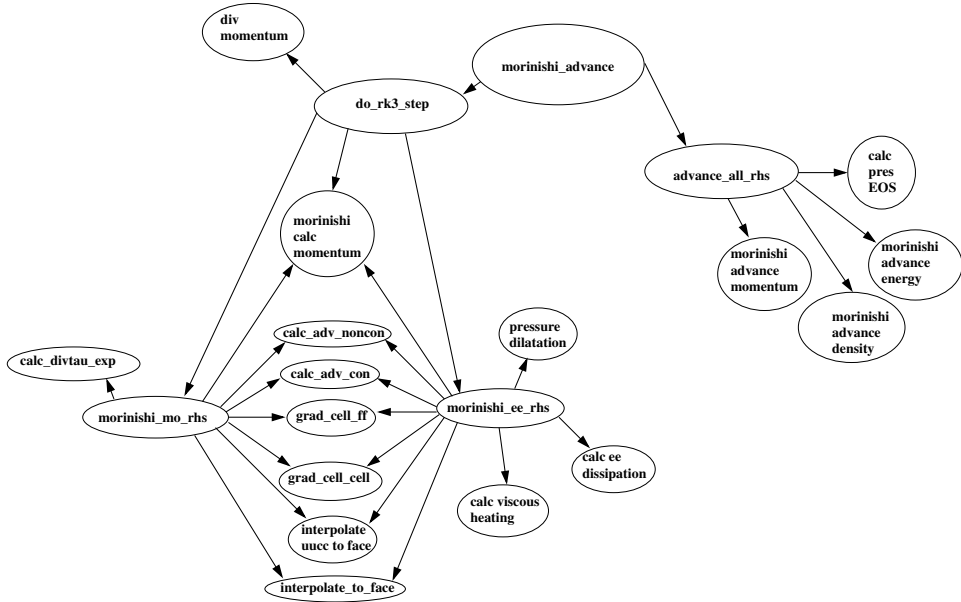


Figure 4.2: Logical map for the compressible flow code.

Number	Name	Ghost Values	Restriction
1	morinishi_calc_momentum		✓
2	div_momentum		✓
3	morinishi_mo_rhs		
4	morinishi_ee_rhs		
5	calc_divtau_exp		
6	grad_cell_ff		
7	grad_cell_cell		
8	calc_adv_con		
9	calc_adv_noncon		
10	calce_kinetic_viscosity		
11	interpolate_uucc_to_face		✓
12	interpolate_to_face		✓
13	grad_cell_eeflux		
14	calc_ee_dissipation		
15	pressure_dilatation		
16	advance_all_rhs		
17	morinishi_advance_momentum	✓	✓
18	morinishi_advance_density	✓	✓
19	morinishi_advance_energy	✓	✓
20	calc_pres_EOS	✓	✓
21	advance_morinishi_coll		
22	do_rk3_step		

Table 4.1: All functions in the code, in no particular order. Check marks indicate the extension to ghost values or restriction from fine-to-coarse mesh values.

Chapter 5

Interface Implementation

The current interface implementation is based on the work of Uddin and Pantano. The method uses a least squares approximation to project fluid (“wet”) values to the fluid-solid interface. Then, a PDE is solved to extrapolate the values on the fluid-solid interface into the solid domain to fill the “dry” cell values. The PDE is elliptic and solved only in the solid, so that the wet values on the interface act as the boundary conditions for the Boundary Value Problem (BVP). The PDE has the property that values are extended orthogonally inward from the boundary and diffusion is added far from the boundary to prevent the formation of sharp gradients.

Because the problem domain (i.e. dry cells) is not necessarily rectangular, a mesh-free method for solving the PDE is implemented. For all dry cells, the stencil is computed and stored, so that the “action” of the matrix can be computed. The discretized linear system is then solved using an iterative linear solver, such as a Jacobian Preconditioned Biconjugate Gradient solver or a Gauss-Seidel method, which requires only the action of the matrix rather than any information about structure. The following contains details about the implementation of the method, as well as selected mathematical details about the method. The code can be found in `interface.f90` and `mpi_interface.f90`.

LS_TYPE	Description
-1	No Solid
1	Circle, center = (0.5, 0.5), radius = 0.1
2	Circle, center = (<i>xxblob</i> , <i>yblob</i>), radius = <i>radblob</i>
3	NACA00XX airfoil, XX= <i>radblob</i> , length = 1
4	Stationary flat plate, thickness= <i>thkblob</i> , length = <i>radblob</i>
5	Pitching, rigid flat plate, thickness= <i>thkblob</i> , length = <i>radblob</i>
6	Deforming flat plate, thickness= <i>thkblob</i> , length = <i>radblob</i>
7	Deforming flat plate with tackle block, thickness= <i>thkblob</i> , length = <i>radblob</i> , block dim=1
8	Cavity, depth= <i>thkblob</i> , length = <i>radblob</i>

Table 5.1: Comparison of runtime (sec), level-set error, and reconstruction error using the “cheap” level-set method and the bisection-based level-set method.

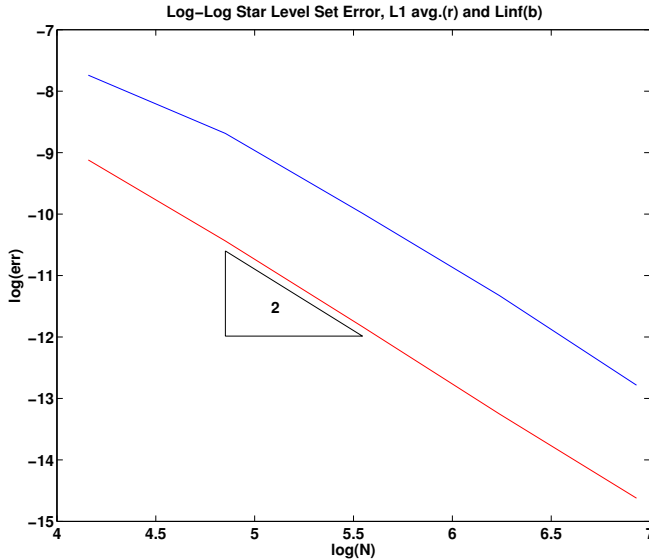


Figure 5.1: Error in zero level-curve computation of the star function.

5.1 Interface Detection

The interface is detected as the zero level-curve of some level-set function ϕ . It is assumed that an accurate level set function is *given* to my code. This level-set function can be an exact function or some approximation to the distance in the vicinity of the interface. The interface is currently specified using a linear approximation the level-set function. This gives second order accuracy in the location of the interface (5.1). Here, level set error is taken to be $|\phi_{exact}|$, for the “star” level-set function

$$\phi(x, y) = \sqrt{\left(x - \frac{1}{2}\right)^2 + \left(y - \frac{1}{2}\right)^2} - (R + \delta \sin(5\theta(x, y))). \quad (5.1)$$

The parameters are taken to be $R = 0.25$ and $\delta = 0.05$. Detection of the interface, distance from a dry cell to the interface, and orientation of the cut relative to the dry cell centroid are all shown in Fig. 5.2. Higher-order level-set detection methods exist, but will not be necessary unless a higher-order accurate level-set function is provided. This is *not* considered to be in the scope of my portion of the project.

Let Γ be the location of the interface, and let ϕ be the corresponding level-set function. Then, we will define the advective velocity a as

$$a \equiv \text{sgn}(\phi) \frac{\partial \phi}{\partial x}. \quad (5.2)$$

Additionally, we will define the sign of a as $s \equiv \text{sgn}(a)$. Then, the level-set function is approximated as

$$\phi(x) = \phi(x_i) + (x - x_i) \frac{\partial \phi}{\partial x}. \quad (5.3)$$

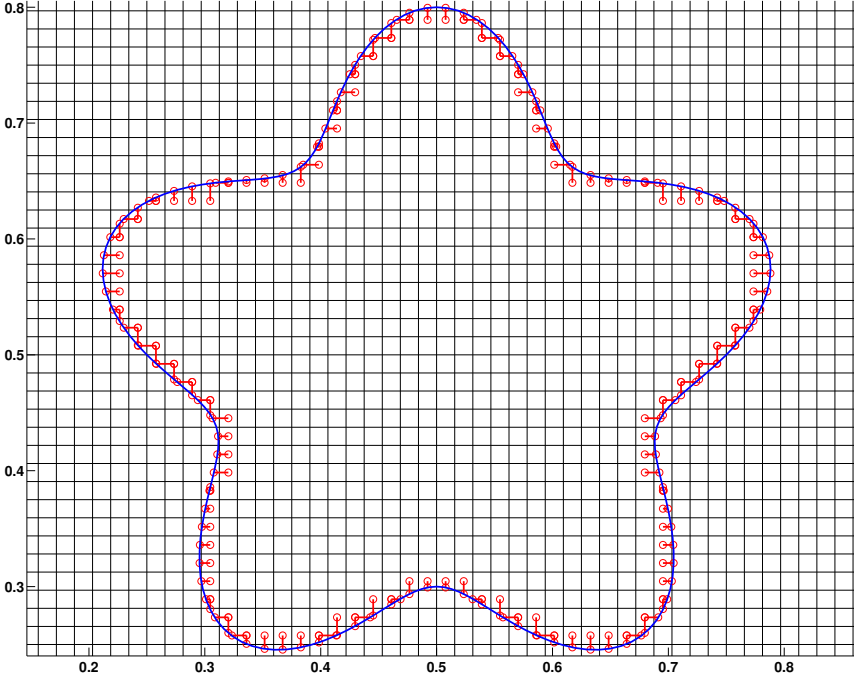


Figure 5.2: Zero level-curve of the star function. Cut cells are denoted with red circles. Location of and orientation to the interface are shown in red. Resolution is $N=64^2$.

On the interface, $\phi = 0$, so the length fraction ℓ occupied by the solid can be calculated as

$$\mu = -\frac{\phi(x_i)}{\Delta x \frac{\partial \phi}{\partial x}}. \quad (5.4)$$

Assume that the interface has an intercept in the range $[x_{i-1}, x_{i+1}]$. The derivative is approximated as

$$\frac{\partial \phi}{\partial x} \approx s \frac{\phi_i - \phi_{i-s}}{\Delta x}, \quad (5.5)$$

and the length fraction is approximated as

$$\mu = \frac{s\phi_i}{\phi_{i-s} - \phi_i} \quad (5.6)$$

It must be noted that it is possible for a cell to be cut by the interface between both $[x_{i-1}, x_i]$ and $[x_i, x_{i+1}]$. In this case, the features of the structure is under-resolved in cell x_i . Such doubly-cut cells can occur at the trailing edge of an airfoil, where the curves denoting the top and bottom edges meet at a single point. This case requires special care, as (5.6) will fail to accurately compute the location. This is due to the fact that the level-set function computes the *shortest* distance from a point to the interface.

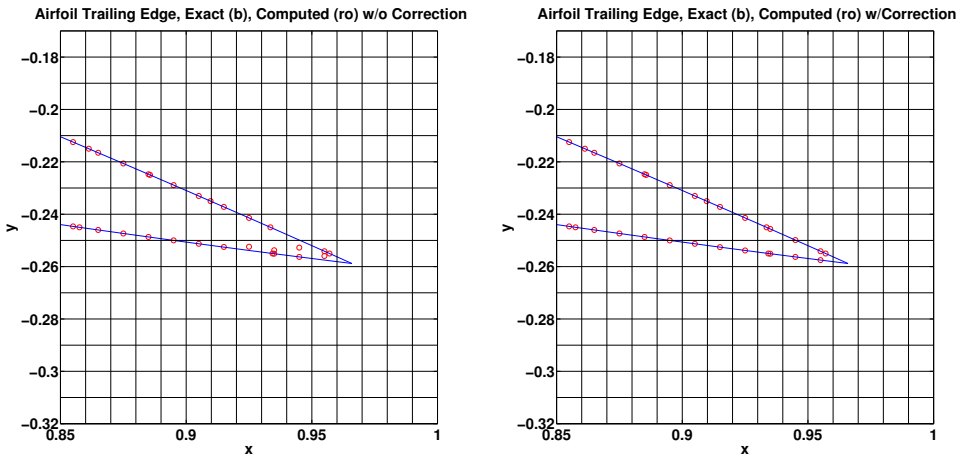


Figure 5.3: Trailing edge of an airfoil without (left) and with (right) the root-finding correction in (5.7). The blue line shows the exact interface for a NACA0012 airfoil with a 15° angle of attack, and the red circles show the computed roots of the level-set function. Chord length is $C = 1$, with mesh spacing $\Delta x = C/100$. The roots are incorrectly predicted by the naive formula (5.6) when the trailing edge is under-resolved by the mesh, but accurately captured using (5.7).

Now, suppose Γ_1 and Γ_2 denote the top and bottom edges of an airfoil respectively, and suppose that $|\phi_{\Gamma_1}(x_i)| < |\phi_{\Gamma_2}(x_i)|$. The ϕ_i will accurately predict the distance from x_i to Γ_1 while under-predicting the distance to Γ_2 . This under-prediction causes (5.6) to inaccurately calculate the location of the interface. A simple correction can be applied if a doubly-cut cell is detected by replacing (5.6) with

$$\mu = 1 - \frac{\phi_{i\pm 1}}{\phi_{i\pm 2s} - \phi_{i\pm s}}, \quad (5.7)$$

where the sign is taken as positive for the intercept in $[x_i, x_{i+1}]$ and negative for the intercept in $[x_{i-1}, x_i]$. Note that calculation of neither intercept uses the value ϕ_i , as it is not known *a priori* which intercept is more accurately predicted by value ϕ_i . See Fig. 5.3

5.2 Boundary Extrapolation Method (BEM)

We wish to extend some field β inward from some closed curve Γ defined as the zero level-curve of some function ϕ . We shall form an approximation of the field as

$$\beta(\phi) = \beta_0 + \beta_1\phi + \beta_2\phi^2. \quad (5.8)$$

Note, this assumes that the field β is well-approximated inside the solid as a polynomial in terms of the level-set function ϕ . For some general fluid flow, this is not necessarily true for the entire region enclosed by the solid. However, close to the interface, it is likely a good approximation for a high-enough polynomial degree.

Define the sgn function, such that

$$sgn(\phi) = \begin{cases} 1, & \phi > 0 \\ -1, & \phi < 0 \\ 0, & \text{otherwise.} \end{cases} \quad (5.9)$$

Then, we can construct the PDE for each value β_k as

$$sgn(\phi) (\nabla \phi \cdot \nabla \beta_k) = \epsilon h^m \nabla \cdot \left(\left| \frac{\phi}{h} \right|^{k+1} \nabla \beta_k \right) \quad (5.10)$$

The parameters ϵ and m are chosen by experiment to be $\epsilon = 1$ and $m = 2$. The value of ϵ determines the strength of the diffusion. A larger value of ϵ corresponds to greater diffusion. Increasing diffusion does not have a large effect on the overall accuracy of the BEM within a two cell band about the interface, although it increases the condition number of the PDE (5.10). This has an effect of increasing the time required to solve the linear system. The parameter m is chosen to ensure that the scaling is regularized with respect to grid scaling.

5.2.1 PDE Discretization

The advective terms are discretized using upwinding relative to the level-set. For each cardinal direction, the following scheme is used,

$$a \frac{\partial \beta}{\partial x} \approx a^+ \frac{\beta_i - \beta_{i-1}}{\Delta x} + a^- \frac{\beta_{i+1} - \beta_i}{\Delta x} = |a| \frac{\beta_i - \beta_{i-s}}{\Delta x} \quad (5.11)$$

$$s = sgn(a) \quad (5.12)$$

Relating (5.11) to (5.10), we find that the advective velocity is related to the level-set function as

$$a = sgn(\phi) \frac{\partial \phi}{\partial x}. \quad (5.13)$$

Here, $a^+ = (a + |a|)/2$ and $a^- = (a - |a|)/2$. Recalling the convention that the level-set function is positive in the liquid and negative in the solid, we find the gradient of the function β is discretized upwind, relative to material flowing into the solid from the interface. Note that this discretization cannot be used adjacent to the interface, as it reaches outside of the region of dry cells. In cells adjacent to the interface, the stencil is modified.

$$\frac{\partial \beta}{\partial x} \approx s \frac{\beta_i - \beta_{\Gamma^*}}{|\eta| \Delta x} \quad (5.14)$$

Here, β_{Γ^*} is the value of the field β approximated on the boundary, in the given cardinal direction. Value $\eta = \frac{x_{\Gamma^*} - x_i}{\Delta x}$ is the signed distance fraction from the cell centroid to the interface. Note that the inherent assumption is that a cell centroid that lies exactly on the interface is considered to be wet, rather than dry. Considering such a cell to be in the solid would lead to a division by zero in (5.14).

A similar discretization is performed for the diffusion stencil of a cut cell. Suppose that a given cell is cut on both sides in the x-direction. Then, the diffusion operator is discretized as

$$\frac{\partial}{\partial x} \left(w \frac{\partial \beta}{\partial x} \right) \approx \frac{r_+ (\beta_{\Gamma^+} - \beta_i) - r_- (\beta_i - \beta_{\Gamma^-})}{\Delta x^2 |\eta_+ \eta_-|} \quad (5.15)$$

$$r_+ = \frac{|\eta_-|(w_+ + w_-)}{\eta_+ - \eta_-} \quad (5.16)$$

$$r_- = \frac{\eta_+(w_+ + w_-)}{\eta_+ - \eta_-} \quad (5.17)$$

If the cell is not cut between x_i and x_{i+1} , set $w_+ = w_{i+1}$ and $\eta_+ = 1$. Similarly, If the cell is not cut between x_i and x_{i-1} , set $w_- = w_{i-1}$ and $\eta_- = -1$. It should be noted that when the cell is not cut, the standard three-point diffusion operator with a variable diffusion coefficient is recovered.

$$\frac{\partial}{\partial x} \left(w \frac{\partial \beta}{\partial x} \right) \approx \frac{(w_{i+1} + w_i)(\beta_{i+1} - \beta_i) + (w_i + w_{i-1})(\beta_i - \beta_{i-1})}{2\Delta x^2} \quad (5.18)$$

5.3 Implementation

Implementation is spread across a number of functions, each with a specific task. The implementation mimics that of the Matlab version of the Uddin BEM.

5.3.1 build_xg_list

Inputs the level set function as a multifab on any number of levels (though currently only a single level), mesh layout, and mesh spacing. Outputs the list of vertices (xlist); the cut-length for each cut (cutlen); the (xi,yj) index of the cut cell (cutidx); the direction of the interfacial cut (cutdir); the number of cuts in each FAB in a single array (sizelist); and the total number of cuts (count_total).

This subroutine approximates the total number of cuts as follows. Let N denote the total number of cells on the mesh level. The number of cut cells is the approximated as $4\pi N^{1/2}$. This supposes that the mesh domain is a square and the interface is a circle with radius $N^{1/2}$. A safety factor of 2 is multiplied to allow for cuts in multiple directions for each cell. Arrays of equal size, summing to the total number of allocated values, are distributed to each FAB. To enhance fault tolerance, each FAB detects at runtime whether or not enough space is available for all cut information. If not enough space is available, the arrays are reallocated in the worker subroutine `build_xg_list_Nd`. The only exit failure condition is if the total number of cells is exceeded in the driver function.

Note: this function allocates variables which are used in other functions. These should be deallocated after the projection coefficients are calculated.

5.3.2 interp_to_gamma

Inputs the level set values on the mesh level (phi); function values on the mesh level (ff); vertex list; cut indices; size list; and total cut count. Output the projection values f_Γ on the boundary for each cut. Currently, outputs three arrays, f_{Γ_0} , f_{Γ_1} , and f_{Γ_2} .

Neighbors are gathered for a least squares reconstruction of the function on the interface (5.19). For the first pass, all wet cells in a 5×5 neighborhood of the dry cut cell are tested. Note that at least 6 values are needed for the least squares reconstruction in 2D (1 function value, 2 gradient values, 3 Hessian values). The number will be higher in 3D. I examine a large area because one may assume that at least half of the cells in the stencil will be dry. After all wet neighbors in the region are found, the Singular Value Decomposition (SVD) is performed, and the rank of the system is determined. If the rectangular matrix is found to be rank-deficient, then the routine attempts to recover. The outermost ring of cells in the 7×7 neighborhood of the cell of interest are tested, and

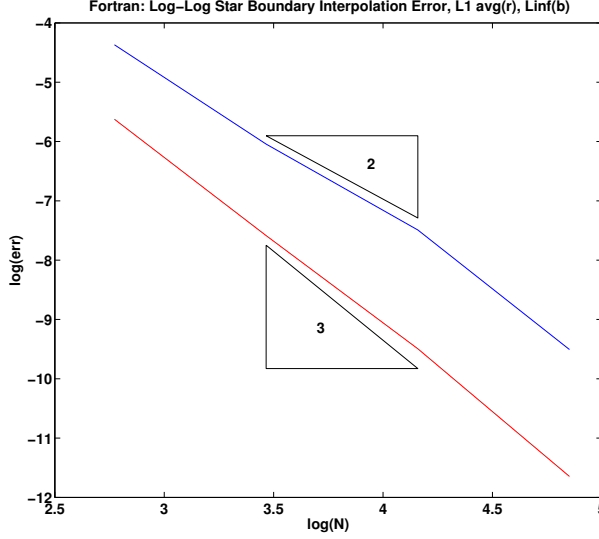


Figure 5.4: Projection error of function (5.20) to the star-shaped interface.

any wet cells are collected. The SVD is then performed again. If at either stage the matrix had sufficient rank, then the LLS problem is solved and the necessary values are reconstructed. If the matrix is still rank deficient, then the deficient eigenvalues are filled with the smallest eigenvalue and the fictitious problem is solved. In its current state, this line has not yet been invoked, as the 5×5 search has always yielded a full-rank matrix.

$$\begin{pmatrix} 1 & \delta x_1 & \delta y_1 & \frac{1}{2}\delta x_1^2 & \delta x_1\delta y_1 & \frac{1}{2}\delta y_1^2 \\ 1 & \delta x_2 & \delta y_2 & \frac{1}{2}\delta x_2^2 & \delta x_2\delta y_2 & \frac{1}{2}\delta y_2^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \delta x_n & \delta y_n & \frac{1}{2}\delta x_n^2 & \delta x_n\delta y_n & \frac{1}{2}\delta y_n^2 \end{pmatrix} \begin{pmatrix} f_\Gamma \\ \frac{df}{dx} \\ \frac{df}{dy} \\ \frac{d^2f}{dx^2} \\ \frac{d^2f}{dxdy} \\ \frac{d^2f}{dy^2} \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix} \quad (5.19)$$

Convergence rate for this subroutine is approximately second order for the star-shaped level-set function, with the function (5.20) taken to be a composition of sines and cosines (Fig. 5.4).

$$f(x, y) = \pi^{-1} (\sin(2\pi x + 1/3)\cos(2\pi y + 1/4) + \sin(2\pi x + 1/3)) \quad (5.20)$$

5.3.3 interp_vel_to_gamma

Inputs the level set values on the mesh level (phi); function values on the mesh level (ff); vertex list; cut indices; size list; and total cut count. Output the projection values f_Γ on the boundary for each

cut. It is assumed that f_{Γ_0} is an input, as velocity is specified on the surface. Only f_{Γ_1} and f_{Γ_2} are output.

Because we assume that velocity is known on the surface, the known value f_{Γ_0} is moved to the right-hand side of the LLS procedure (5.21).

$$\begin{pmatrix} \delta x_1 & \delta y_1 & \frac{1}{2}\delta x_1^2 & \delta x_1\delta y_1 & \frac{1}{2}\delta y_1^2 \\ \delta x_2 & \delta y_2 & \frac{1}{2}\delta x_2^2 & \delta x_2\delta y_2 & \frac{1}{2}\delta y_2^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \delta x_n & \delta y_n & \frac{1}{2}\delta x_n^2 & \delta x_n\delta y_n & \frac{1}{2}\delta y_n^2 \end{pmatrix} \begin{pmatrix} \frac{df}{dx} \\ \frac{df}{dy} \\ \frac{d^2f}{dx^2} \\ \frac{d^2f}{dxdy} \\ \frac{d^2f}{dy^2} \end{pmatrix} = \begin{pmatrix} f_1 - f_\Gamma \\ f_2 - f_\Gamma \\ \vdots \\ f_n - f_\Gamma \end{pmatrix} \quad (5.21)$$

It should be noted that both `interp_to_gamma` and `interp_vel_to_gamma` require the LAPACK library, as it has efficient subroutines. The original implementation used an SVD to determine if the system was singular, with an additional subroutine to compute the Moore-Penrose pseudo-inverse of the system. However, it was found to be approximately 25% faster to use LAPACK's least-square solver, which also computes the rank of the system.

5.3.4 `interp_ml_box_to_gamma`

A more sophisticated implementation of the routine that interpolates function values to the interface Γ . This subroutine accepts the same arguments as `interp_to_gamma`. However, it grabs all function values on level *reclv* within the 7×7 box surrounding the cut cell. The original subroutine only grabbed all points within the 7×7 box that were in the same FAB as the cut cell. This caused problems, depending on how the multiFAB cut the solid. This version grabs all valid grid cells on the given level. However, it *does not* grab ghost cells. This is to prevent poor reconstructions at or near coarse/fine mesh boundaries. Near mesh boundaries, it is possible that not enough data can be collected in the normal direction to obtain a sufficiently accurate approximation to $\frac{\partial f}{\partial n}$ on the interface. This failure to approximate the normal derivative of the function leads to a failure to accurately extrapolate the fluid values into the solid at coarse/fine mesh boundaries. In even benign flow conditions, this can lead to failure of the fluid algorithm due to negative density, energy, or diffusion coefficients.

To prevent this, we detect the case when the 7×7 stencil cannot be filled using grid cells (again, not ghost cells). When this is detected, we default to a reconstruction using coarse mesh values with the same 7×7 stencil on the coarse mesh. The proper nesting of the adaptive grid hierarchy ensures that the coarse mesh level will be able to entirely fill the stencil for the least-squares reconstruction. See Fig 5.5 The use of coarse mesh values in the reconstruction of solid values on the fine mesh is assumed to be acceptable as it occurs *only* near the coarse/fine mesh boundary. Note that on the coarse side of the mesh boundary, only coarse values will be used. While this is not a smooth transition from using the fine values to using a much larger spatial stencil with the coarse values, it is a practical and “easily” implemented solution. In practice, this works well in the flow conditions with multiple levels of mesh refinement across the solid. To accomplish the gathering of boxes across multiple FAB's, it is first detected whether or not the 7×7 box intersects a FAB. If so, the intersection box is passed to the subroutine `gather_subbox_2d` and the values are stored in the data structure.

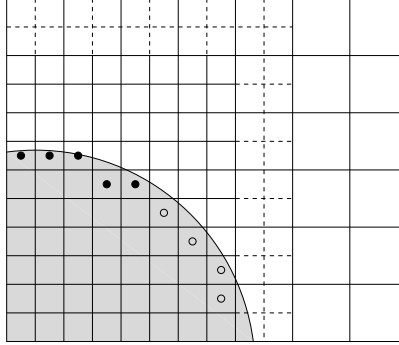


Figure 5.5: A solid structure (shaded) cutting an adaptive grid. Cut cells (circles) use fine values (filled) or coarse values (open) for the least-squares reconstruction, depending on their proximity to the coarse/fine mesh boundary. Only cells with the surrounding 7×7 stencil in the fine grid use fine grid cells for the reconstruction. Ghost cells are outlined with dotted lines.

Note: A similar implementation exists for *interp_ml_vel_box_to_gamma*, which uses fine level values where available, and defaults to using coarse level values when not enough fine data exists.

5.3.5 detect_box_fluid_path

Attempts to detect whether a “fluid path” exists from the cut cell to a cell gathered to use in the least squares reconstruction of the function value and normal derivative. A fluid path is defined as some path under the “taxicab metric” that starts at the cut cell and ends at the cell to be used in the least squares reconstruction. If a fluid path exists from the cut cell to the cell in question, that cell may be used in the least squares reconstruction; otherwise, the cell is not used in the reconstruction. This prevents the least squares reconstruction from using cells on the wrong side of the wing in the section near the tail. However, it will not prevent the LSR from gathering all such points, but only those cut cells that are not cut by a “filament,” i.e. cut on opposite sides in the same direction. Such a structure is pathological, and detection/prevention is not practical. We *can* prevent the method from gathering data in cases where there are two points across the wing. This method utilizes the “box” data structures generated by *interp_ml_box_to_gamma*.

5.3.6 uddin_proj_coeffs

Calculates the coefficients for the linear system defined by the boundary extrapolation PDE (5.10). The parameters are typically taken to be $\epsilon = 1$ and $m = 2$ so that the correction terms are of the same order. Note that the level set value ϕ acts as the diffusion coefficient. It is worth pointing out that as the solid is better resolved, the linear system will become more stiff. Thorough testing has shown that for cells with $2.5\Delta x$ of the fluid/solid interface Γ , the diffusion parameter has little effect on the accuracy of the extrapolation in the context of the fluid algorithm. The parameter was tested from values $\epsilon \in [1^{-10}, 1000]$. Because the diffusion is dependent on distance from the interface, this is not unreasonable. It is possible that the method can be simplified, as the vast majority

of the run-time is occupied by the extrapolation of fluid values into the solid.

$$\text{sgn}(\phi) (\nabla \phi \cdot \nabla \beta_k) = \epsilon h^m \nabla \cdot \left(\left| \frac{\phi}{h} \right|^{k+1} \nabla \beta_k \right) \quad (5.22)$$

The equation (5.10) is discretized using centered differencing for the diffusive terms and upwinding for the advective terms. The notion of the “upwind” direction is taken from the gradient of the level-set function. This subroutine produces the matrix coefficients (organized as a grid-based multiFAB with $2 \cdot \text{DIM} + 1$ components) and the forcing function associated with the system. The forcing function is non-zero only for cells such that the boundary cuts between the cell and at least one of its neighbors. Both the matrix and the forcing function are zero in all cells such that $\phi \geq 0$, which indicates that the cell is “wet” and therefore not in the solid. The current implementation does not support cells that are doubly cut in a single direction (e.g. cut on the high and low sides in the x-direction). This will require special detection. For more details on the method, see Uddin and Pantano (2014).

5.3.7 uddin_force

Because the matrix of a given order will be the same for all $3 + \text{DIM}$ state variables, we only need to compute it once per Runge-Kutta stage. However, the forcing function will be different for each state variable. As such, this function repeats some of the functionality of `uddin_proj_coeffs` by calculating the forcing function alone for a given input state variable.

5.3.8 pbicgstab_grid

Applies a Bi-conjugate Gradient Stabilized linear solver with Jacobian preconditioning for the linear system given by the input “matrix” `upc` and the forcing function. An initial guess is provided by the user to start the iterative solver. The level-set function `phival` must be provided. This is used for error checking to determine where the linear operator must be applied. This subroutine calls procedures `apply_upc_op` and `apply_upc_jpop` which apply the linear operator and the Jacobian preconditioning operator, respectively.

While inputs `upc` and `force` do not have ghost values, the input `guess` should have at least one ghost value. Because the linear operator is applied uniformly throughout the domain, at the edge of a FAB, the stencil will reach outside of the subdomain. The subroutine `pbicgstab_grid` also uses the `saxpy` operators native to `BoxLib`.

5.3.9 bpm_gs_grid_solver

On a single grid, successively applies a Gauss-Seidel operator until convergence. Convergence is calculated inside the worker subroutine and passed to this master subroutine. Note that after each iteration, it is necessary to call `multifab_fill_boundary` to fill cells between grids of the same level of mesh refinement. It is not necessary to fill coarse/fine grid cells (which act as boundary conditions to a given mesh level if the grid crosses the interface), as these are assumed to be filled upon input to this subroutine and are not updated on each iteration.

5.3.10 apply_bpm_gs_op

Uses a Gauss Seidel-style operator to update grid values. The new and old grid values are passed to the function. The grid is scanned sequentially from the lower-left to the upper-right corner, with perviously updated values used in computational stencil.

$$f_{i,j}^{n+1} = G(f_{i,j-1}^{n+1}, f_{i-1,j}^{n+1}, f_{i,j}^n, f_{i+1,j}^n, f_{i,j+1}^n) \quad (5.23)$$

In this subroutine, in addition to the newly updated state variable being passed out, the difference between the new and old state variables is calculated. This allows us to test for convergence for the “master” routine that calls this “worker” routine.

5.3.11 recon_overwrite_fo

Uses a “first-order” overwrite of the field value with the extrapolated value of the fluid in the solid. Calculates the reconstruction

$$\beta_{recon} = \beta_0 + \phi\beta_1. \quad (5.24)$$

It was found through experiment that the inclusion of the second order term $\phi^2\beta_2$ did little to increase accuracy in physics simulations, due to high-sensitivity to perturbations. One can obtain a constant reconstruction (rather than a linear reconstruction) by setting $\beta_1 \equiv 0$.

5.3.12 bpm_reconstruct_all

Function wrapper to reconstruct all state variables in the solid, collecting a number of calls. Forms and solves the linear systems for all state variables up to first order. Once the values β_0 and β_1 have been computed, the solution inside the solid is overwritten. This subroutine is called *before* each call to calculate the RHS of the Runge-Kutta stage, and one more time at the end. The final reconstruction is performed to obtain an accurate face velocity at the fluid/structure interface for calculation of the time step. Currently, all field values are reconstructed, while only velocity is needed. I have not yet written a wrapper for a single variable.

5.3.13 ml_bpm_recon_all

Function wrapper to extrapolate all state values from the fluid into the solid. Build the necessary data structures (`build_xg_list`) and calculates the necessary boundary conditions (`interp_ml_box_to_gamma` or `interp_ml_vel_box_to_gamma`). The linear operators are formed only once per order per level (`uddin_proj_coeffs`); after the operator is formed on a level, only the forcing function must be uniquely calculated for each state variable. It should be noted that the forcing function does depend on the value of the state variables, but the linear operator only depends on the geometry (which does not change between state variables). The values for density, energy, pressure, and velocity are reconstructed on each level. After all calculations have been completed on each mesh level, the grid data structures, boundary values, linear operators, and work variables are deallocated.

5.3.14 bpm_reconstruct_var_ilev

Performs all operations to reconstruct a state variable on a given level of mesh refinement, with the appropriate ghost values filled if the grid cuts across the fluid/structure interface. It should be

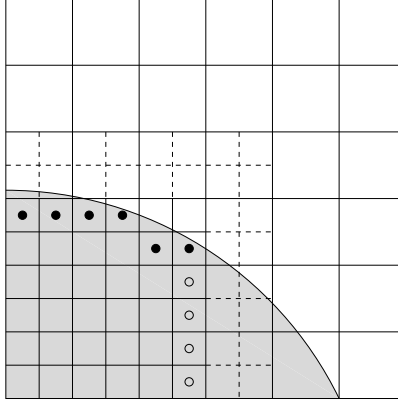


Figure 5.6: A solid structure (shaded) cutting an adaptive grid. Cut grid cells (filled circles) have a neighboring cell in the liquid (white). Ghost cells adjacent to the fine grid are outlined with dotted lines. Fine level cells on the coarse/fine boundary (open circles) require ghost values filled from the underlying coarse grid.

noted that if the grid level boundary cuts the fluid/structure interface, special care must be taken in solving the linear system for the normal derivative to the surface. In calculating the state variable value on the interface, the value is already available; it is stored and passed as an argument. The normal derivative of the state variable is not calculated as part of the fluid algorithm. However, if the grid boundary crosses the interface, ghost values of $\frac{\partial f}{\partial n}$ are required as “boundary conditions” in solving the linear system. As such, the value of

$$\frac{\partial f}{\partial n} = \nabla f \cdot \hat{n} = \nabla f \cdot \nabla \phi \quad (5.25)$$

is computed on the current level and the next coarsest level, then all ghost cells are filled. Here, let ϕ be the level-set function. Then, the linear system may be solved for f_0 and $\frac{\partial f}{\partial n}$, using either BICG or the Gauss-Seidel method. Finally, the state variable in the solid is reconstructed as

$$f = f_0 + \frac{\partial f}{\partial n} \phi. \quad (5.26)$$

It should be noted that if more than one level of mesh refinement crosses the fluid/solid interface, there will be duplication of efforts in the calculation of $\frac{\partial f}{\partial n}$ on the current level and the next coarsest level. I am trading efficiency for a clean, self-contained implementation. Additionally, the subroutine `calc_gradf_dot_gradphi` uses the level-set value to calculate the surface normal. If the level-set function is inaccurate (i.e. we use a cheap, less-accurate method of calculating the level-set or we are using data near the trailing edge, where the surface is under-resolved), this function won’t be as accurate. A better implementation for the NACA problem could use the explicitly calculated surface normal.

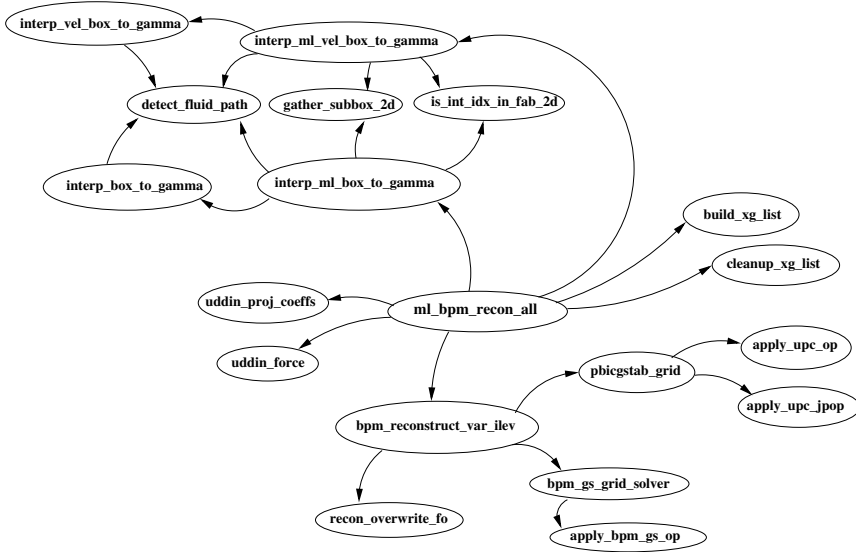


Figure 5.7: A map of the functions written for the interaction of the flow with the interface. Arrows point from the caller to the callee. The fluid algorithm *only* calls `ml_bem_recon_all`, which calls the rest of the necessary functions. Reconstructions for *all* state variables on *all relevant* levels are performed from here.

5.3.15 apply_sponge_target

Applies a “sponge” region that attempts to drive the solution to the “target” value. Values `sp_coeff` and `sp_exp` determine how aggressively the solution is driven to the target. Input `sp_dir` determines the direction in which the sponge is applied (1 = x, 2 = y, 3 = z). The values `sp_lo` and `sp_hi` determine the bounds of the range over which the sponge is applied. The logical variable `sp_out` is true on the high side of the domain and false on the low side of the domain. This alters the behavior of the filter so that stronger filtering is applied near the physical boundary, regardless of whether that was on the high or low side of the sponge domain.

$$\hat{\phi}_{i,j} = \phi_{i,j} - \sigma \left(\frac{1}{8} (\phi_{i+1,j} + \phi_{i-1,j} + 4\phi_{i,j} + \phi_{i,j+1} + \phi_{i,j-1}) - \phi^T \right) \quad (5.27)$$

$$\sigma = \text{sp_coeff} \left(\frac{x - \text{sp_lo}}{\text{sp_hi} - \text{sp_lo}} \right)^{\text{sp_exp}} \quad (5.28)$$

Currently, the field is changed in place, meaning that advanced values can use already filtered values in calculating the filter, similar to Gauss-Seidel filtering. This may or may not be the best way this to do, but it damps waves very quickly, and it avoids the need for another grid variable.

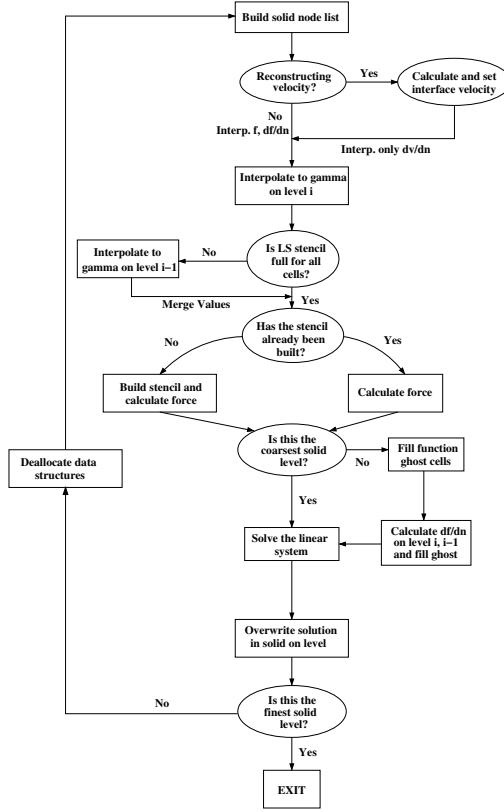


Figure 5.8: Flow chart for the execution of the solid wrapper function `ml_bpm_recon_all`.

5.4 NACA-00XX Level-Set Function

A set of subroutines designed to allow for the simulation of a symmetric NACA airfoil are included in the file `naca00tt.f90`. Additionally, special routines and calls are included in the interface module to allow for the user to specify an angle-of-attack for the NACA airfoil, and to accurately handle the thin trailing edge. Let us assume that the airfoil has unit chord-length, and that the airfoil oriented at a zero degree angle-of-attack is the “standard” frame of reference. Let us refer to the airfoil at any non-zero angle-of-attack as being in the rotated frame of reference.

5.4.1 `naca00tt_fxn2d`

Equation of the top or bottom edge of a symmetric NACA airfoil in the standard frame of reference. For error checking purposes, it is assumed that the input x-location is greater than or equal to the location of the leading edge (also input). If not, the code will fail. The coefficients are *slightly* modified from the typical coefficients so that the top and bottom edges meet at a single point. This

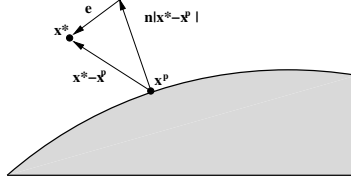


Figure 5.9: Illustration of the objective function, i.e. the magnitude of vector $\mathbf{e}$.

keeps the level-set function that defines the solid from being poorly defined.

5.4.2 `naca00tt_tanvec2d`

The tangent vector to the upper or lower edge of a symmetric NACA airfoil. Can be rotated (based on whether the point is on the top or bottom of the airfoil) to find the normal vector to the surface. The coefficients have the same modification as in Sec. 5.4.1 such that the top and bottom edges meet at a point.

5.4.3 `ls_objective`

Objective function for the level-set method that uses a Newton iteration. To calculate a level-set function, one must find the shortest distance from the point-of-interest to the surface (e.g. the orthogonal projection of the point-of-interest onto the surface). The objective function is non-negative and zero at the orthogonal projection onto the surface. Let x^* be the point-of-interest and x^p be the proposed orthogonal projection. Then,

$$f_{obj}(x^*, x^p) = |(x^* - x^p) - \hat{n}(x^p) |x^* - x^p|| = |e|, \quad (5.29)$$

where $\hat{n}(x^p)$ is the normal vector to the airfoil surface at point x^p .

5.4.4 `rotata_point_aoa`

Rotates an input point around the location of the leading edge by a give angle-of-attack. All points are rotated into the standard frame of reference *before* the level-set function is calculated, so that *no* level-set computation occurs in the rotated frame of reference. Note that the title is correct; there was a typo in the code that I copied and pasted too many times to fix before I found it.

5.4.5 `level_set_naca00tt`

Computes the level-set function for a single point $xvec$ for a NACA airfoil, where tt is the height as a fraction of chord length. Inputs specify the maximum number of iterations and the minimum acceptable error tolerance. The input point should be rotated *before* input in there is a non-zero angle-of-attack. This is an expensive method for calculating the level-set function, as it attempts a Newton method for minimizing the objective function that appears in Sec. 5.4.3. For points such that $x \gg$ than the trailing edge, there may not exist an orthogonal projection of the point-of-interest onto the surface. In that case, the trailing edge point may be taken as the minimizer of the objective function. It is also worth noting that at the leading edge, the problem is very stiff, as the normal

direction changes rapidly for small perturbations in x . Often, the leading edge is spuriously found to be the minimizer of the objective function. This could potentially be remedied by parametrizing the surface in terms of arclength instead of as a function of x or having an adaptive step-size used in computing the derivative in the Newton minimization routine. However, this would introduce its own fun set of problems. It doesn't appear to greatly affect the accuracy of the root-finding, which is only secondary to the real purpose: computing physics at the fluid/structure interface. Unless problems arise later, we will assume that this is an acceptable compromise between accuracy and efficiency.

5.4.6 `level_set_naca00tt_bisect`

Computes the level-set function for a single point $xvec$ for a NACA airfoil, where tt is the height as a fraction of chord length. The input point should be rotated *before* input in there is a non-zero angle-of-attack. This is a cheaper alternative to `level_set_naca00tt`, as this method uses the bisection method rather than Newton's method. Inputs specify the maximum number of iterations and the minimum acceptable error tolerance. While Newton's method may have a theoretically faster convergence rate, the bisection method is resilient to the stiffness that plagues Newton's method. The bisection method is posed as a root-finding problem, rather than an error-minimization problem as with Newton's method, so a different objective function must be used.

$$f_{obj} = (x^* - x^p + \hat{n}(x^p)|x^* - x^p|) \cdot t(x^p) \quad (5.30)$$

Let the vector $t(x^p)$ be the vector that is tangent to the airfoil at the point x^p . Note that the vector $x^* - x^p + \hat{n}(x^p)|x^* - x^p|$ is the "error" vector mentioned in Sec. 5.4.3. When the point x^p is the orthogonal projection of x^* onto the interface, (5.30) is zero. The function will take positive and negative values, depending on the orientation between vectors e and $t(x^p)$.

It is important to note that while this is a valid approach in 2D, there is no equivalent method to bisection in 3D, so this will not apply to 3D flows. For 2D flows, though, the bisection method is much faster than using Newton's method.

5.4.7 `level_set_naca00tt_cheap`

Computes the level-set function for a single point $xvec$ for a NACA airfoil, where tt is the height as a fraction of chord length. The input point should be rotated *before* input in there is a non-zero angle-of-attack. This is the cheapest level-set method we have, as it bases level-set value on the vertical distance between the point-of-interest and the airfoil interface. This works well everywhere on the wing, except near the leading edge, which has an arbitrarily large dy/dx . This method is an *order of magnitude* cheaper than even the bisection method, which is an order of magnitude faster than Newton's method.

Table (5.2) compares runtime, level-set error (measured as the average $|y_{comp} - y_{exact}|$), and $\mathcal{L}^1$ average reconstruction error for a sinusoidal function using the BEM with the "cheap" level-set method and the bisection-based level-set method. In general, the cheap method is an order of magnitude faster than the bisection method, but the level-set function error is larger. However, we are not interested in the level-set function for its own sake, but rather for extrapolating fluid values into the solid. Under this metric, the two methods yield results within 1% of one another, although the "cheap" method has larger error.

AMR Levels	Cheap			Bisect.		
	Time	ϕ Error	Rec. Err.	Time	ϕ Error	Rec. Err.
0	4.1e-3	2.2e-5	5.9e-3	8.4e-2	3.2e-5	5.9e-3
1	1.8e-2	3.5e-5	2.3e-3	4.1e-1	1.4e-5	2.3e-3
2	7.5e-2	1.4e-5	1.1e-3	1.7	5.8e-6	1.1e-3

Table 5.2: Comparison of runtime (sec), level-set error, and reconstruction error using the “cheap” level-set method and the bisection-based level-set method.

Chapter 6

MPI Details

6.1 Function Hooks

BoxLib include a number of “hooks” (or more properly *interfaces* in Fortran-ese) that simplify the calls to standard MPI functions. This eliminates the common arguments such as the communicator, data length (calculated automatically), MPI data type, etc. You can still make the proper MPI function calls, but this simplifies things. The following are common MPI hooks that are frequently used in the code.

- **bl_error(str)**: Halt all threads and print an optional message in **str**. Should be used in a parallel code rather than the typical Fortran call to **exit**. Functionally, it appears to be equivalent to **parallel_abort**.
- **parallel_abort(str)**: Halt all threads and print an optional message in **str**. Should be used in a parallel code rather than the typical Fortran call to **exit**.
- **parallel_barrier()**: Calls MPI.Barrier. Causes all processors to wait until all processors have reached the barrier. Note: It is very easy to deadlock your code using barrier calls improperly. Make sure that *all* processors can actually call the barrier.
- **parallel_bcast(data, root=sendproc)**: Blocking call to broadcast data from one processor (root) to all other processors. The documentation says that the “root” argument is optional, but execution is unpredictable. Note that *all* processors must call the broadcast function, not just the root processor. It is very easy to deadlock your code when using this improperly.
- **parallel_reduce(global, local, MPI_OP)**: Calls MPI.AllReduce to gather all local values to a global value, using some MPI operator. Operators include MPI.SUM and MPI.MAX. The Fortran interface automatically detects the data type and calls the proper hook for the data type.
- **parallel_send(dat, dest, tag)**: Blocking send call for multi-dimensional arrays. Halts execution on the processor until the matching receive call has executed. Sends data to the specified processor with the specified tag. Fortran interface automatically detects the data type and calls the proper hook.
- **parallel_send.i(dat, dest, tag)**: Blocking send call for a single integer. Added by Jemison for dynamic array size communication.

- `parallel_send_l(dat, dest, tag)`: Blocking send call for a single logical. Added by Jemison to allow processors to communicate whether or not they have found a value or finished a task.
- `parallel_recv(dat, source, tag)`: Blocking receive call for multi-dimensional arrays. Halts execution on the processor until data is received from the “source” processor with the given data tag.
- `parallel_recv_i(dat, source, tag)`: Blocking receive call for integer. Halts execution on the processor until data is received from the “source” processor with the given data tag. Added by Jemison.
- `parallel_recv_l(dat, source, tag)`: Blocking receive call for logical. Halts execution on the processor until data is received from the “source” processor with the given data tag. Added by Jemison.

6.2 MPI Interface

The implementation for the multi-processor interface code has been separated from the single-processor functions, mainly for the sake of the programmer’s sanity. However, there are substantial differences in the complexity and flow-of-control of the single- and multi-processor versions.

Recall that the `interp_box` functions were developed in light of unacceptable error when interpolating flow variables to the interface Γ using only the values in the FAB (Fortran Array Box) in which the cut-cell was located. Instead, we consider some $M \times M$ stencil about the cut cell and gather all cells in all FAB’s and on all processors that lie in the stencil. Thus, unless we lie at a coarse/fine mesh boundary, the stencil will be full. The velocity interpolation subroutines are similar to the state variable subroutines, with the exception that velocity is specified at the interface, thereby changing the linear system that must be solved. Since we are interested in the MPI-related details, only the state-variable interpolation subroutines will be discussed. However, the flow-of-control of the velocity subroutines is identical.

6.2.1 Subroutine `mpi_interp_box_to_gamma`

We are interpolating a state variable and normal derivative to the interface, as computed by the level-set function. Each processor holds all of the information that was held globally in the single-processor subroutine. This includes ...

- Interface cut location coordinates (ordered by FAB’s).
- Cut cell indices (ordered by FAB’s).
- Number of cut cells on the processor.
- Number of cut cells per FAB.

The structure of the subroutine is similar to the single-processor version: on each processor, scan over all FAB’s; on each FAB, scan over all cut cells; gather all data in the $M \times M$ stencil for the cut cell; interpolate values to the interface. However, the gather phase may require interprocessor communication. To allow communication, each processor must call the same communicator function, acting as either master (if data is needed in that processor’s stencil) or worker (if data is needed

in some other processor's stencil). Four distinct control situations must be accounted for in this subroutine.

1. All processors have the same number of FAB's and cut cells. This is the (unlikely) ideal which requires no special accomodation.
2. Processors have the same number of FAB's but a different number of cut cells.
3. Processors all have FAB's, but some processors have different number of FAB's.
4. One or more processors has no FAB's.

To account for these situations, we require the use of 4 flags, which exist on each processor.

1. `comps_done_local`: Denotes if all computations on all FAB's are finished on the local processor.
2. `comps_done_global`: Denotes if all computations on all FAB's finished on all processors.
3. `proc_done`: Denotes if all computations are done on a *single* FAB on a single processor.
4. `all_procs_done`: Denotes if all computations are done on a *single* FAB on all processors.

To ensure that all processors have access to the necessary data, I use the following control algorithm.

1. On each processor, start work on FAB j .
2. If no FAB's left on processor i , set local values to `.true.` for `comps_done_local` and `proc_done`. Enter a loop
 - (a) Look for a processor trying to communicate. If the processor exists, distribute data as needed.
 - (b) Use logical reduce on `proc_done` $\rightarrow$ `all_procs_done` and `comps_done_local` $\rightarrow$ `comps_done_global`.
 - (c) Check if `all_procs_done == .true..`
 - (d) Check if `comps_done_global == .true.`
 - (e) Exit loop if `all_procs_done`. Else, repeat loop.
3. Do work on all cut cells in FAB j
4. If finished with all cut cells on FAB j , enter a loop.
 - (a) Look for a processor trying to communicate. If the processor exists, distribute data as needed.
 - (b) Use logical reduce on `proc_done` $\rightarrow$ `all_procs_done` and `comps_done_local` $\rightarrow$ `comps_done_global`.
 - (c) Check if `all_procs_done == .true..`
 - (d) Check if `comps_done_global == .true.`
 - (e) Exit loop if `all_procs_done`. Else, repeat loop.

In this way, we loop over all data in “lockstep” across all processors. Each processor works on one cut cell, and does not progress to the next cell until all processors are finished with that cell. Similarly, each processor works on one FAB and does not progress to the next FAB until all processors are finished with that FAB. This is not the most efficient control structure, but it was the simplest to devise (note: “simplest to devise” should not be taken to mean “simple to devise”).

Note: there are actually three “loop check” control structures. One for if there are no FAB’s on a processor (as the FAB loop would exit without executing), which also executes once a processor is done with all of its FAB’s and nodes; one if we had FAB’s but no nodes on the current FAB; and one for if we had FAB’s and nodes but have completed work on all nodes of the current FAB. It is worth noting that once all FAB’s have been completed on a processor, the condition for exiting the “loop check” is whether or not all computations are done globally, whereas if we are simply finished with all work on a single FAB, we exit once all processors are done with their current FAB.

6.2.2 Subroutine `mpi_gatherall_subbox_caller_2d`

The 2D communicator to gather data in the $M \times M$ stencil between processors. Each processor enters with a box (perhaps empty) and a flag denoting if the processor needs to gather stencil data. If the processor needs data, then it will be a master at some point. If not, the processor will only act as a worker. This subroutine loops over each processor sequentially, broadcasting whether or not the current processor needs data. If not, nothing is done. If data is needed on the current processor, the current processor is master and all others are workers. The box range is sent to all processors, which gather function and level-set data into an 1D array. The worker tells the master if it has data to send. If the worker has data to send, the 1D array is sent back to the master processor. The master processor receives the data flag. If there is data, the 1D array is reshaped and loaded into the stencil. This subroutine is called whenever the processors are required to communicate in Section 6.2.1.

6.2.3 Subroutine `mpi_interp_ml_box_to_gamma`

The intent and basic control structure of this subroutine is the same as that in Section 6.2.1, interpolating values to an interface within a multilevel hierarchy. However, for cut cells that lie near a coarse/fine mesh boundary, we use the `mpi_interp_box_to_gamma` subroutine on the next coarsest mesh level. The current implementation is far from optimal, but appears to give results consistent with single-processor runs, and is consistent in both $\mathcal{L}^1$ and $\mathcal{L}^\infty$ error when using between 2-8 processors. The current implementation computes the coarse interpolation *for all* cut cell nodes, then attempts to compute the interpolation for all nodes on the fine mesh level, excluding ghost cells. For any cell with an under-filled $M \times M$ LSQ stencil on the fine mesh level, the coarse level interpolation is used. Note that to exclude ghost cells from the gather operation, one uses the `get_box` or `get_ibox`

The procedure for computing the interpolation on the fine mesh level is identical to that in Section 6.2.1, including the communication loops and processor work flags. However, communicating the data to the coarse mesh level is significantly more complicated. There are essentially 6 stages to the procedure.

1. Communicate all node data to all processors, stored in “global” variables. Create a reverse-lookup table.

2. Compute cut-cell data on the coarse mesh.
3. Determine which data lives on which processor. Distribute relevant entries in the reverse-lookup table.
4. On each processor, eliminate data from the “global” structures which do not live on the processor.
5. Perform the interpolation procedure on the cut cells that live on the processor on the coarse mesh level.
6. Gather coarse data to processors, using the reverse-lookup tables to correlate the coarse-level reconstruction with fine-level data.

A legitimate question is why do we have to go through all of this trouble? Why can't we just compute the interpolation on the coarse level directly using the data from the fine level that lives on the same processor? One finds that the domain in physical space that lies on Processor N on Level I is not necessarily the same domain in physical space that lies on Processor N on Level I+1. For this reason, if processor data is passed directly from the fine level to the coarse level, there may be cut cells passed that do not reside on the same processor on the coarse level. This would result in failure of the interpolation algorithm. An additional complication is that data on the coarse processor must be collected such that data on the same processor and in the same FAB must be sequential when input to subroutine `mpi_interp_box_to_gamma`. So, step 2 must scan the global list and locate not only the coarse-level processor, but also the coarse-level FAB that contains the cut cell in question.

Thus, we must collect all fine-level cut cells across all processors and distribute them to the appropriate processor on the coarse mesh level. The reverse-lookup table provides us a straight-forward way to gather the unordered data from coarse-level processors and return them to their original indices on the fine-level processors.

6.2.4 Reverse-Lookup Table

The reverse-lookup table is generated during Step 1 in Section 6.2.3, during the gathering of data into global arrays. The table consists of the processor rank and index of the cut-cell, physical coordinate, and f_Γ within the 1D input array to the subroutine `mpi_interp_ml_box_to_gamma`. This data is global and is distributed to processors on the coarse mesh level in the same way as cut-cell data. In Step 6 of Sec. 6.2.3, when transferring data from coarse-level processors to fine-level processors, we simply scan over processors, gather data from all workers that corresponds to the master processor, and send the data and the index. Once the data is received by the master from the worker, the index from the reverse-lookup table is used to put the f_Γ and $\left.\frac{\partial f}{\partial n}\right|_\Gamma$ in the correct array locations.

6.3 MPI FAQ's

Why does my new subroutine bind up when using MPI?

Each working subroutine has the same basic format. First, we loop over levels. On each level, we loop over FAB's and do work. After the end of the FAB loop, we fill the boundary values between FAB's of the same level and (if necessary) fill physical boundary values (i.e. domain boundary ghost cells).

```
do n = 1,nlevs
  do i = 1,nfabs(data(n))          ! number of Fortran Array Boxes
    ptr => dataptr(data(n),i)
    lo = lwb(get_box(data(n),i))
    hi = upb(get_box(data(n),i))
    select case(dim)
    case(2)
      call ccfunction(ptr(:, :, 1, :), lo, hi, numghost, dx(n, :))
    case(3)
      call ccfunction(ptr(:, :, :, :), lo, hi, numghost, dx(n, :))
    end select
  end do

  call multifab_fill_boundary(data(n)) ! fill ghost cells BETWEEN fabs at same level
  call multifab_physbc(data(n),1,1,numcomp,bc(n)) ! fill ghost cells at physical
                                                    ! boundary (wall, inflow, slip) only

end do

do n=nlevs,2,-1 ! restrict, starting at finest and working down
  ! Synchronizes coarse data underlying a fine mesh.
  call ml_cc_restriction(data(n-1),data(n),rr(n-1))

  ! Fills coarse/fine ghost cells
  call multifab_fill_ghost_cells(data(n),data(n-1),numghost,rr(n-1,:), &
    bc(n-1),bc(n),1,1,numcomp)
end do
```

If you move the `multifab_fill_boundary` call *inside* the FAB loop, there is a barrier call that may or may not be reached by all processors, depending on your mesh layout. This can cause your code to deadlock intermittently. This is because all processors may not have the same number of FAB's, so the barrier function won't be called the same number of times. This is not the case in the *level* loop, as each processor has the same number of levels allocated (regardless of whether or not any data lives there), so the synchronization cannot cause deadlock. It should be noted that some of LBNL's example codes (not intended for MPI use) use the incorrect function format, synchronizing boundary cells in the FAB loop rather than the *level* loop. So, if you are copying and pasting a function "template," this can cause you problems.